Trabajo de Grado

TRIO': O como ahora las tautologías son tautologías y el infinito es infinito.

Autor

Isaac Carlos Turquie.

Director

Dr. Miguel Felder.

A Leti, Kari y mis Viejos.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todos aquellos que contribuyeron, de una u otra manera a la mejora de este trabajo, entre los que se destacan Guillermo Diaz, Marité Guardarucci, Nelly Echebest, Marcelo Rabey, Cristina Vacchino y Lorenzo Zivano. En particular quisiera agradecer a Ricardo Rosenfeld por su esmerada lectura y útiles sugerencias. También a mi director Miguel Felder por todo su tiempo dedicado.

El autor.

ABSTRACT

La creciente utilización de sistemas de tiempo real en un amplio campo de nuestra vida moderna que requieren un alto grado de confiabilidad, hace necesario el uso de técnicas de verificación formal de los mismos.

Se define en este trabajo *TRIO'*, como una extensión de la lógica temporal lineal de primer orden TRIO, donde se incorpora una semántica más natural y adecuada para modelar sistemas de tiempo real y probar propiedades sobre los mismos, pudiendo expresar, además, el tiempo en forma infinita con la ayuda de variables definidas para tal efecto.

La factibilidad de los algoritmos de análisis de TRIO' se demuestra con la implementación de los algoritmos de Generación de Modelos y de History-Checking, con un funcionamiento decidible.

Palabras claves: sintaxis, semántica , modelos, Generación de Modelos, History-Checking, decibilidad.

ÍNDICE

INTRODUCCIÓN

Los sistemas de tiempo real son aquellos cuyas funcionalidades están restringidas por estrictas respuestas en función del tiempo, o sea, que no solo importa la sucesión de acciones a través del tiempo, sino también en que momento se ejecutan y cuanto tardan en hacerlo. Son claves en algunas áreas (manejo de procesos industriales, torres de control de aviones, monitoreo de pacientes, etc.) donde una falla no solo puede causar daños materiales, sino también la perdida de vidas humanas, por lo que la corrección de los mismos juega un rol fundamental. Otros ejemplos son los protocolos de comunicación (en particular para multimedia y en sistemas de tratamiento de información dinámica donde se hace necesario bajas demoras en la comunicación). La mayoría de estos sistemas mantienen una interacción constante con su entorno y más que devolver un valor en su terminación algunos nunca terminan.

Una de las soluciones es el uso de especificaciones formales y su posterior ejecución, lo que permite chequear donde tales especificaciones cumplen, o no, con las propiedades esperadas del sistema, en una etapa temprana del desarrollo. Esto se podría realizar sin necesidad de tener implementado, y eventualmente funcionando, el sistema, evitando el elevado costo de los errores que, como se dijo en el párrafo anterior, se podrían generar.

Existen métodos semi-formales para la especificación, el diseño y la verificación, pero ofrecen límites en cuanto a sus capacidades de análisis debido a que su semántica ha sido definida informalmente. Por otro lado los métodos formales, desarrollados hasta el momento no pueden trabajar en forma decidible cuando intentan representar el tiempo en forma infinita.

Una característica importante de los sistemas de tiempo real es que poseen una naturaleza event-driven más que data-driven por lo que el uso de especificaciones funcionales no sería el más adecuado, en particular para expresar precedencia y/o requerimientos de tiempo con lo que se encuentra a la lógica temporal la candidata más apropiada debido a su expresibilidad y naturalidad para especificar el tiempo.

Pnueli [Pn77] propuso el uso de la lógica temporal para describir y verificar programas concurrentes, con la que se pueden describir la ocurrencia de eventos y propiedades en el tiempo (por ej.: precedencia, invarianza, fairness, exclusión mutua, no deadlock) como fórmulas de tal

lógica. La lógica temporal es una clase de la lógica modal[I] donde los operadores representan el tiempo, como por ejemplo: Always (siempre) o su dual Eventually, o Sometimes, (alguna vez), Until [GPSS80], (el cual puede ser definido en función del Always, o viceversa), Next, Since (como análogo al Until) [LPZ83]; las lógicas branching, como se detalla más adelante, agregan cuantificadores (por ejemplo: $\forall$ y $\exists$Until) para expresar distintos caminos (computaciones). Muchos ejemplos de especificaciones hechas usando lógica temporal pueden encontrarse en [Pn77], [MP 81], [La83] y [CE81] entre otros.

En los sistemas de tiempo real la corrección y funcionamiento depende, entre otras cosas, de la capacidad de indicar momentos precisos en que ocurre cierto evento o propiedad, por lo tanto, el uso de la lógica temporal tradicional no sirve, debido a que sus operadores no pueden medir el tiempo en forma cuantificada entre una aserción y otra; por ejemplo se puede expresar que una propiedad suceda después de un evento dado, pero representar que suceda luego de una cantidad exacta de unidades de tiempo después puede resultar exageradamente complicado. Con este último objetivo fue definido TRIO [GMN89], como una extensión de la lógica temporal de primer orden junto con el operador temporal Dist (A,t) que indica que la propiedad A tomara lugar exactamente en t unidades de tiempo antes o después del momento actual, dependiendo si t es negativo o positivo respectivamente. De este operador se pueden derivar Futr(A,t) y Past(A,t)[II], como abreviaturas del Dist, pero correspondiendo al futuro y al pasado respectivamente. A modo de ejemplo, se puede decir que TRIO fue usado como lenguaje de especificación para sistemas de tiempo real, como lenguaje de descripción de hardware de alto nivel y en la descripción del control de una estación de potencia del ENEL (Ente Nacional de la Energía Eléctrica de Italia). Además de ser usado como el kernel de un completo entorno de especificación de sistemas de tiempo real [Mor89]. Sin embargo, los métodos definidos hasta ahora para analizar semánticamente las fórmulas hace que se presenten resultados no naturales o que se presenten posibles contradicciones, como se verá más adelante. Es por eso que se decide estudiar una nueva definición semántica para TRIO. De esta investigación, junto con otras necesidades luego explicadas, surge ***TRIO'***.

Dentro de la lógica temporal existen dos grandes grupos, según la representación del conjunto de estados y sus transiciones. Uno, las lógicas lineales, en el que esta representación es una secuencia de estados que expresa todas las posibles computaciones, donde el tiempo es lineal, es decir existe un único futuro para cada estado. El otro grupo, las lógicas branching, en el que la representación de los

[I] Más información sobre la lógica modal se puede encontrar en [HC68].

[II] Los primeros usos de la lógica temporal en la computación en los que estaba el operador Futr, no incluian al Past, hasta que su uso fue justificado en [LPZ85].

estados tiene una estructura de árbol, donde existen diferentes caminos a seguir desde un momento dado y cada camino expresa los distintos estados que puede alcanzar un programa, por lo que se puede, entonces, usar los cuantificadores $\forall$ y $\exists$ para indicar estos diferentes caminos.

La potencia expresiva de cada grupo es distinta y ninguno contiene al otro, por ejemplo cierta clase de fairness no puede ser expresada en la lógica branching y por el otro lado, la cuantificación de los caminos no se puede expresar en la lógica lineal, por ejemplo para describir programas no determinísticos, como pueden ser algunos programas concurrentes[III]. Un argumento en favor de la lógica branching ha sido su mejor eficiencia para la verificación automática, y en cambio, a favor de la lógica lineal está su simplicidad para expresar propiedades y que en realidad la mayoría de las cosas que se quieren probar, en gran parte de los sistemas de tiempo real, son sobre todas las computaciones posibles, sin necesidad de discriminación alguna. CTL* es un formulismo que combina ambos grupos [EH86]. En la actualidad, hay en la comunidad científica creyentes y detractores para cada uno de los grupos.

Con respecto a los métodos formales para sistemas de tiempo real, se pueden distinguir:

- ☑ los operacionales (los que describen el sistema como funciones u operaciones) o
- ☑ los descriptivos, como en este trabajo, (donde el sistema se describe enumerando sus propiedades).

En [ADC90] y [HNSY94] se encuentra una herramienta en la que se combinan las dos formas de descripción, donde se especifica el programa, a través de grafos temporizados, los que son esencialmente autómatas extendidos con variables reales llamadas relojes, y con fórmulas TCTL (Timed Computation Tree Logic) se describen las propiedades que se quieren probar sobre el mismo. Además existe una herramienta, llamada KRONOS, en la que se puede modelizar y verificar en forma automática, la que implementa el algoritmo de model-checking simbólico presentado en [HNSY94]. El model-checking simbólico es una técnica para determinar los estados que satisfacen una fórmula a través de un análisis del espacio de estados en forma simbólica (más que enumerativa) y obteniendo los conjuntos de estados que satisfacen una fórmula como un punto fijo de un funcional sobre un conjunto de estados. Otro ejemplo son las Time Basic nets (TBnets) [GMMP91], las que son una extensión de las redes de Petri, donde cada token está asociado con una marca que representa el instante en el cual fue creado por un disparo y cada transición está asociada con una

[III] En la lógica lineal un programa concurrente es expresado como el interleaving de sus estados, por lo que expresar algo del tipo: 'La propiedad siempre P vale siempre, al menos, a lo largo de una ejecución del programa' es casi imposible.

función que describe la relación entre las marcas de los tokens removidos por un disparo y las marcas de los tokens creados por el mismo disparo. Existe, también, una herramienta, llamada CABERNET, la que permite especificar y verificar sistemas con este tipo de redes. RTTL (Real-Time Temporal Logic) es un formalismo que está organizado en dos capas, una consistente en autómata de estados extendido y la otra compuesta por fórmulas de la lógica temporal de primer orden clásica, las que hacen referencia a eventos y estados, donde un evento indica el cambio de valor de verdad de un predicado a medida que avanza el tiempo. Las propiedades de tiempo real deben ser hechas introduciendo un reloj al autómata, o sea agregar una variable discreta que se incrementa monótonamente a medida que el sistema evoluciona, esto ocasiona la posibilidad de perder la noción del tiempo como implícito del sistema, llevando al especificador la responsabilidad del control del tiempo como por ejemplo para que los estados no avance si el tiempo no fue modificado.

En este trabajo, se utilizará una lógica temporal lineal de primer orden para expresar el programa y sus propiedades en forma descriptiva.

A grandes rasgos, cuando se expresa el sistema en forma descriptiva, se puede trabajar con verificación formal de dos formas distintas. Dada una especificación (Γ), que describa los requerimientos del sistema, expresada como una fórmula bien formada, se considera que las propiedades que debe cumplir el mismo (Π), expresadas también como una fórmula, como correctas para (o que cumplen con) Γ , si :

I. en forma deductiva: se amplia el sistema formal con las fórmulas de la especificación como axiomas y vale si la fórmula de las propiedades es un teorema de este sistema utilizando reglas de decisión. Esta es generalmente una tarea manual tediosa y donde juega un rol importante la intuición del usuario. Además la mayoría de las lógicas temporales son incompletas, por lo que habría fórmulas que son verdaderas en ciertos sistemas y no son teoremas.

II. en forma semántica: se intenta buscar una interpretación, según la definición semántica de esta lógica, que sea modelo (una interpretación que hace a la fórmula verdadera) de $\Pi \wedge \Gamma$.

Esta última aproximación representa los valores fisicos alcanzados por una evolución o *historia* del sistema como una interpretación del lenguaje utilizado, de manera tal de considerarla como un posible estado de la implementación si la misma es un modelo de la fórmula que representa la especificación y las propiedades del sistema.

Como dijo anteriormente, si se lograra ejecutar una especificación expresada con una lógica temporal y se pudiera observar el funcionamiento del sistema, se podría detectar errores en una etapa temprana del desarrollo. El problema se alcanza, en todos los trabajos presentados hasta el momento, al intentar trabajar con una estructura temporal subyacente correspondiente a un dominio infinito, con lo cual muchas de las propiedades que uno desea probar hacen al sistema indecidible. Por ejemplo, al buscar un modelo para la especificación dada, pero en el caso de un dominio infinito no se podría garantizar la terminación de un algoritmo de búsqueda que enumere todos los modelos posibles. Este ha sido otro de los objetivos de TRIO', el cual ha sido alcanzado, debido a la nueva semántica definida y a los algoritmos desarrollados.

El uso de ventanas finitas pueden ser vistos como una aproximación del dominio infinito, donde se puede chequear que el sistema cumple con las propiedades deseadas e inferir que su funcionamiento va a ser similar en el caso infinito. Este procedimiento, como en el caso de testing donde no se asegura la ausencia de errores, no puede afirmar la corrección en el caso infinito. Sin embargo se puede hacer una analogía entre esta aproximación y la aritmética usada en todas las computadoras, ya que asumimos que todos los resultados aritméticos obtenidos son válidos, a menos que se produzca overflow.

La dificultad principal es como evaluar las fórmulas que caen afuera de este intervalo finito.

Una de las soluciones ha sido el determinar un valor 'por default' a todas estas fórmulas. Esta solución da a las fórmulas un valor antinatural, por ejemplo con el valor False al darle a las tautologías, que caen afuera del intervalo finito, o análogamente con el valor True y las contradicciones.

Otra idea [MGG92] propone tener en cuenta solo los valores de las variables de la fórmula de manera tal que se evalúe siempre dentro de la ventana finita determinando un sorte para cada dominio, con lo que hace su uso limitado al intervalo obtenido de las variables en cuestión, llegando posiblemente a no poder evaluar la fórmula.

Por último, considerar el dominio temporal finito como circular, donde la aritmética sobre los valores temporales está definida de manera tal que cualquier referencia hacia un valor fuera del intervalo finito, es redireccionada dentro del mismo en forma circular. Por ejemplo con un dominio entero, se puede utilizar una aritmética modulo k para la suma o resta de los valores del intervalo finito, donde k es el tamaño del dominio. Esta solución solo es considerable para sistemas periódicos.

Como hemos visto, en la mayoría de los casos no se puede asegurar el valor de verdad natural de muchas fórmulas, ni siquiera para las tautologías y las contradicciones. En este trabajo, como ya se

ha mencionado, se ha definido una nueva semántica, entre otras razones, para tratar de mantener este valor de verdad de la mayoría de las fórmulas, o al menos, como se demostrará, de las tautologías y las contradicciones. Además se buscó que la nueva semántica de TRIO' permita no solo especificar el valor de verdad de una fórmula atómica fuera del intervalo finito, sino también evaluar cualquier otra fórmula. Incluso se logró que se pueda utilizar variables destacadas, cuyos dominios pueden ser infinitos, y discretos[IV], para hacer referencia sobre propiedades que pueden darse en cualquier instante de tiempo del dominio temporal infinito.

Se ha dicho, que la idea de la verificación es buscar modelos (que representan una historia del sistema) para una fórmula (que lo describe) expresada en una lógica temporal cuya semántica permite dominios infinitos. Uno de los algoritmos desarrollados, el de búsqueda de modelos, conocido como *generación de modelos*, permite buscar satisfabilidad, es decir saber si una fórmula dada tiene, o no, al menos un modelo, lo que se traduce como si es, o no, implementable. El mismo algoritmo se puede utilizar como generador de casos de simulación, del cual podemos obtener eventuales historias del sistema y para probar propiedades sobre una especificación. El otro algoritmo, trabaja con una interpretación y una fórmula dada y su tarea es chequear la interpretación con la fórmula para saber si es, o no, un modelo de la misma, es decir si es, o no, un estado posible de alcanzar por el sistema que se ha implementado con la fórmula dada. Este último algoritmo es llamado *history-checking*[V].

Una técnica de validación se definió [MMG92] por implementar la semántica de TRIO a través del Tableaux Method [Sm68], el cual provee un intérprete abstracto del lenguaje. Este algoritmo está implementados en [FM94]. La terminación esta garantizada bajo la hipótesis de que todos los dominios sean finitos. Este método es ampliamente usado en la lógica temporal para verificar la satisfabilidad de una fórmula constructivamente [BPM83, Wo83] y para derivar implementaciones desde modelos de una especificación [CE81].

Los algoritmos definidos en este trabajo han tomado como base el Tableaux Method, pero describiendo los dominios de los modelos de manera simbólica para trabajar con los dominios infinitos y mantener aún la decibilidad.

No solo se han desarrollado para ambos algoritmos una demostración formal de su funcionamiento, sino también se ha implementado un prototipo para cada uno, demostrando efectivamente su factibilidad.

[IV] Una mayor discusión sobre granularidad se encuentra en Trabajos Relacionados.

[V] Estos algoritmos son análogos a los de model-cheking en la lógica branching [QS81]

Un ejemplo de un sistema en TRIO' está desplegado a lo largo de todo el trabajo. Primeramente se da su especificación y sus propiedades, luego se busca un modelo con el algoritmo de generación de modelos y por último se chequea una fórmula de la especificación contra una historia del mismo con el algoritmo de history-checking.

El resto del trabajo está organizado como sigue: el capítulo 2 define TRIO', dando su semántica y sintaxis y detallando el ejemplo de la especificación antes mencionado. Luego el capítulo 3 describe un conjunto de propiedades para este formalismo. A continuación los capítulos 4 y 5 desarrollan los algoritmos de generación de modelos y de history-checking respectivamente, junto con un ejemplo, la idea de la demostración formal y el cálculo de la complejidad para cada uno. Un pequeño comentario sobre las implementaciones realizadas está en el capítulo 7. Los trabajos relacionados, futuros y las conclusiones finales se describen en el capítulo 8. En el Apéndice A se desarrollan las demostraciones del capítulo 3. Por último, las demostraciones formales de los algoritmos de generación de modelos y de history-checking se detallan en los apéndices B y C.

Se adjunta un anexo, con todos los detalles técnicos de las implementaciones y un disquete con los códigos fuentes y los ejecutables de los prototipos realizados.

FORMALISMO NUEVO: TRIO'

Se define en este trabajo TRIO' como una extensión de la lógica temporal lineal de primer orden con el operador temporal Dist, llamada TRIO [GMM89]. El operador Dist, provee una métrica del tiempo, para expresar la distancia (en tiempo) entre dos propiedades o eventos. El significado de una fórmula depende del instante de tiempo en que se evalúa. Semánticamente se simula el dominio temporal infinito con una ventana finita y se le dará el mismo valor a los predicados cuando sean evaluados en cualquier instante fuera de esta ventana finita. TRIO' incorpora el concepto de variable infinita, cuyo dominio infinito, permite hacer referencia a cualquier instante de tiempo del dominio temporal infinito.

Sintaxis de TRIO'

El alfabeto de TRIO' incluye un conjunto de nombres de variables, funciones y predicados, y un conjunto de símbolos. Las variables y los predicados son divididos en dos grupos: TI (Time Independent) cuyo valor se mantiene constante con el tiempo y TD (Time Dependent) en el que los valores dependen del momento en que se evalúen. Cada variable tiene asociado un tipo o *Dominio* el cual determina los valores que puede llegar a tomar. En este punto debemos aclarar que las variables TD no son tomadas como variables en el sentido clásico (pueden usarse en cuantificadores expresando distintos valores) sino más bien son consideradas como valores constantes en cada instante de tiempo, es decir que solo pueden variar con la modificación del tiempo. Existe, siempre, un dominio distinguido llamado *Dominio Temporal*, el cual es numérico de por sí, discreto e infinito y un subconjunto de este, llamado *Dominio Temporal Finito*, el cual es, como su nombre lo indica, finito. Puede haber variables TI distinguidas, a las que llamaremos infinitas, las que solo podrán ser usadas en el término temporal del *Dist*. Una variable infinita no podrá estar dentro del alcance[VI] de un cuantificador de otra variable infinita[VII]. Todo nombre de función tiene asociada una aridad mayor o igual que 0, en este último caso tenemos lo que llamamos *Constante*, y un tipo para cada elemento del dominio y para la imagen. Las funciones sobre términos con variables infinitas deberán ser lineales.

[VI] Asumimos el alcance de un cuantificador en la forma usual.

[VII] Observar que lo que se pide es que las variables infinitas no estén anidadas.

Una especificación realizada con TRIO' puede usarse para modelar un sistema de tiempo real, donde los componentes físicos y las relaciones invariantes entre ellos pueden ser vistos como constantes y predicados TI, las relaciones temporarias y eventos pueden ser representados como predicados TD, los valores y cantidades físicas que son sujetos a cambios con el tiempo pueden ser vistos como variables TD, las funciones pueden ser usadas para describir operaciones fijas del sistema, las variables TI sirven como términos de los cuantificadores para expresar predicados del sistema y por último las variables infinitas permiten recorrer todos los instantes de tiempo necesarios del sistema.

Si se toman los valores alcanzados por el sistema, en un instante cualquiera, para todos los datos que son representados por predicados y variables se tiene una evolución o historia del sistema.

Los predicados <,<=, = y los demás operadores relacionales sobre números son asumidos en la forma usual además de ser TI, de manera tal de poder ser usados, entre otras cosas, con elementos del dominio temporal. Las funciones de suma y resta son tomados como funciones totales. Todas estas propiedades son asumidas como parte de la especificación sin necesidad de listarlas explícitamente.

Los símbolos del *lenguaje* son:

- *los proposicionales:* $\wedge$ *y* $\neg$,
- *el cuantificador :* $\forall$,
- *y el operador temporal : Dist.*

Se define inductivamente *término* como:

1. toda variable es un término,

2. toda función n-aria lineal aplicada a n términos es un término,

3. toda función n-aria no lineal aplicada a n términos, donde ninguno de los n términos tiene una variable infinita, es un término.

El *tipo* de un término es determinado en la forma usual : Si el término es una variable entonces su tipo es el tipo de la variable. Si el término es una aplicación de una función su tipo es el tipo de la imagen.

Una *fórmula* se define inductivamente como:

i) todo predicado aplicado a n términos es una fórmula (fórmula atómica)

ii) sí A y B son fórmulas entonces $A \wedge B$ *y* $\neg A$ *son fórmulas.*

iii) sí A es una fórmula y x es una variable TI no infinita, entonces $\forall x\ A$ *es una fórmula.*

iv) sí A es una fórmula y t es un término de tipo numérico entonces Dist (A, t) es una fórmula.

v) sí x es una variable infinita y A es una fórmula tal que x solo aparece en los términos temporales del Dist y A no tiene otra variable infinita, entonces $\forall x\ A$ *es una fórmula.*

La definición 3. y v) restringen las variables infinitas a los usos enunciados anteriormente.

La fórmula Dist (A,t) significa intuitivamente que A vale exactamente en t unidades de tiempo, después si t es positivo o antes en caso contrario, con respecto al momento corriente.

Una variable x se define como libre sii x no está dentro del alcance de un cuantificador $\forall x$. Una fórmula A se define como *cerrada* sii ella no contiene ninguna variable TI libre. También definimos la sustitución de la variable x en A por b ($A(^{x}/_{b})$), como el reemplazo de todas las ocurrencias de la variable x por b en A.

De las definiciones anteriores se derivan los operadores $\vee$, $\rightarrow$, $\leftrightarrow$, True, False, etc., y el cuantificador existencial de la forma estándar. Para facilitar la escritura de las fórmulas se puede definir un gran número de operadores temporales, como por ejemplo:

$$Futr\ (A,\ t\ ) \equiv t \geq 0\ \wedge\ Dist\ (A,t)$$

$$Past\ (A,\ t\ ) \equiv\ t \leq 0\ \wedge\ Dist\ (A,t)$$

$$AlwF\ (A) \equiv \forall t\ (Dist\ (A,\ t\ ),\ t > 0$$

$$AlwP\ (A) \equiv\ \forall t\ (Dist\ (A,\ t\ )\ ),\ t < 0$$

$$Always\ (A) \equiv \forall t\ (Dist\ (A,\ t\ )$$

$$SomF\ (A) \equiv\ \neg AlwF\ (\neg A)$$

$$SomP\ (A) \equiv\ \neg AlwP\ (\neg A)$$

$$Sometimes\ (A) \equiv SomP\ (A)\ v\ A\ v\ SomF\ (A) \equiv \exists t\ (Dist(A\ ,\ t\ ))$$

$$Lasts\ (A,t) \equiv \forall t'\ (\ 0< t' < t \rightarrow Futr\ (A,\ t')\ )$$

$$Lasted\ (A,t) \equiv\ \forall t'\ (\ 0< t' < t \rightarrow Past\ (A,\ t')\ )$$

$$Until\ \ (A_1,\ A_2\ ) \equiv \exists t\ (\ t > 0 \wedge Futr\ (A_2,\ t\ ) \wedge Lasts\ (A_1,t\ )\ )$$

$$Since\ (A_1,\ A_2\ ) \equiv \exists t\ (\ t > 0 \wedge Past\ (A_2,\ t\ ) \wedge Lasted\ (A_1,t\ )\ )$$

$$NextTime(A\ ,\ t\ ) \equiv Futr(A,\ t\ ) \wedge Lasts\ (\neg A\ ,\ t\ )$$

$$LastTime(A\ ,\ t\ ) \equiv Past(A,\ t\ ) \wedge Lasted\ (\neg A\ ,\ t\ )$$

$$UpToNow\ (A\ ) \equiv \exists t\ (\ t >0 \wedge Past\ (A,\ t\ ) \wedge Lasted\ (A\ ,\ t\ ))$$

$$Becomes\ (A\ ) \equiv A\ \ \wedge UpToNow\ (\neg A\ )$$

El significado intuitivo de las fórmulas anteriores es para el instante de tiempo corriente: para Futr (A, t) (Past (A, t)) que la fórmula A va a ser True, t unidades de tiempo después (antes) que ahora; AlwF(A) (AlwP(A)), significa que A va a ser True en todos los instantes futuros (pasados) de tiempo; Always (A) dice que A es True en todo instante de tiempo; SomF(A) (SomP(A)): A va a ser True alguna vez en el futuro (pasado); Sometimes (A), indica que A va a ser True alguna vez, en el pasado, presente o futuro; Lasts (A,t) (Lasted(A, t)) implica que en las próximas (pasadas) t unidades de tiempo, A va a ser (fue) True; Until(A_1, A_2) significa que A_2 será True alguna vez en el futuro y hasta entonces vale A_1, análogamente en Since(A_1, A_2) A_2 fue True alguna vez en el pasado y desde entonces A_1 fue True; NextTime (A,t) implica que la próxima vez que A sea True será dentro de t unidades y hasta entonces es False, LastTime(A , t) representa que la última vez que A fue True fue hace t unidades y desde entonces fue False; UpToNow (A) dice que A vale desde alguna vez en el pasado hasta ahora ininterrumpidamente; por último Becomes (A) indica que A vale en el momento actual pero no valió en el pasado inmediato.

Además se pueden derivar de estos operadores los llamados *weak*, en los que no se incluye que el segundo argumento sea verdadero en el Since o en el Until, por ejemplo:

$$UntilW\ \ (A_1,\ A_2\ ) \equiv\ AlwF\ (A_1)\ \ \vee Until\ (A1,A_2\ )\ ,$$

$$SinceW\ (A_1,\ A_2\ ) \equiv\ AlwP\ (A_1\ ) \vee Since\ (A_1,A_2\ )\ ,$$

o los que modifican los intervalos de tiempo en que valen incluyendo, o excluyendo, los límites de los mismos y/o agregando el estado presente, por ejemplo:

$$Lasts'\ (A,t) \equiv\ Last\ (A,\ t\ ) \wedge A\ ,$$

$$Until'\ (A_1,\ A_2\ ) \equiv Until\ (A_1,\ A_2\ ) \vee A_2.$$

$$Since^{p}(A_1, A_2) \equiv Since(A_1, A_2) \vee A_2.$$

Se puede observar que TRIO' además de los operadores de la lógica temporal clásica posee otros, como por ejemplo los operadores que pueden expresar precedencia y distancia entre distintos espacios de tiempo, por lo que su potencia expresiva es al menos tan expresivo como la lógica temporal clásica, además, tiene todos los operadores de TRIO, por lo que las fórmulas del mismo están incluidas en TRIO'. De aquí que podamos derivar todas las implementaciones realizadas en TRIO para trabajar en TRIO' con la nueva representación del tiempo.

Semántica de TRIO'

Se define una interpretación como una 'Temporal Structure', desde la cual se puede obtener la noción de estado, como una asignación de valores a variables y predicados, y de función de evaluación, la cual asigna un valor de verdad para cada fórmula según el instante de tiempo en que se evalúa.

Una temporal stucture S:(D,T,G,L,O) es definida como :

- D es un conjuntos de dominios de variables $D=\{D_1,....,D_n\}$, donde D(x) denota el dominio de x.
  - Si x es una variable infinita, entonces D(x) es un conjunto linealmente ordenado, infinito y discreto.
  - Si x no es una variable infinita entonces D(x) es un conjunto finito.
- T son los dominios temporales. $T=\{ T_f , T\infty \}$
  - $T\infty$ es el dominio temporal infinito. Se supone que es un grupo abeliano, linealmente ordenado, por lo tanto tiene elemento neutro, inverso, una operación +, una relación de equivalencia (que es asociativa, reflexiva, simétrica, transitiva y conmutativa) y la relación de orden <=. $T\infty$ es, además, un conjunto infinito y discreto Ej.: $\mathbb{N}$, $\mathbb{Z}$, etc.
  - T_f es la ventana finita con la que simularemos el dominio infinito, T_f es una cadena de $T\infty$, con elemento máximo y mínimo, por lo que T_f está contenido estrictamente en $T\infty$ y es un conjunto linealmente ordenado.

- G es la parte TI. $G=\{\xi,\Pi,\Phi\}$
  - ξ es una función definida sobre un conjunto de nombres de variables TI tal que, para todo nombre de variable TI x, $\xi(x)$ pertenece a D(x) para algún D(x) que pertenece a D.
  - Π es una función de evaluación definida sobre un conjunto de nombres de predicados tal que, sí p es un predicado r_ario TI, Π asigna una relación a p, tal que $\Pi(p)$ esta contenido en Dp_1 X.....X Dp_r, donde Dp_j pertenece a D, para todo j:[1..r].
  - Φ es una función definida sobre un conjunto de nombres de función tal que si f es una función m-aria, entonces $\Phi(f)$ es una función total del tipo: Df_1 X.....X $Df_m \rightarrow Df_0$ con Df_k pertenece a D para cada k que pertenece a [0..m].

- L es la parte TD de la ventana finita. $L=\{ (\eta_i,\Pi_i)$ tal que i pertenece a $T_f \}$

  Cada par (η_i,Π_i) define el estado de la estructura temporal finita en el instante i del dominio temporal finito.
  - η_i es una función definida sobre variables TD y
  - Π_i es una función definida sobre un conjunto de predicados TD tal que :

    para cada variable TD z, entonces $\eta_i(z)$ pertenece a D(z) (D(z) pertenece a D); y si p es un predicado TD r-ario, $\Pi_i(p)$ está contenido en Dp_1 X.....X Dp_r, donde Dp_j pertenece D, j pertenece a [1..r].

- O es la parte TD del resto del dominio temporal. $O=\{\Pi_e, \eta_e \}$
  - η_e es una función definida sobre variables TD y
  - Π_e es una función definida sobre un conjunto de predicados TD tal que:

    para cada variable TD z, $\eta_e(z)$ pertenece a D(z) (D(z) pertenece a D) y si p es un predicado TD r-ario, $\Pi_e(p)$ está contenido en Dp_1 X.....X Dp_r , donde Dp_j pertenece D, j pertenece a [1..r].

Las funciones Π_i y η_i , cuando $i \in T_f$, definen los valores para los predicados y las variables TD cuando estos son evaluados dentro de la ventana temporal finita. Análogamente las funciones Π_e y η_e, cuando $i \notin T_f$ evalúan cuando se está fuera de esta ventana.

Una temporal structure S = (D, T , G, L) se dice *ADECUADA* para una fórmula TRIO' si D contiene dominios de evaluación correctos para todas las variables que ocurren en ella y si las

funciones ξ, η_i , η_e , Π , Π_i , Π_e y Φ asignan valores del tipo apropiado a todas las variables, predicados (TI y TD) y a todas las funciones.

Sí se puede definir una estructura adecuada, entonces se define una *Función de Evaluación* S_i para saber el valor de verdad de cada fórmula cerrada para la estructura S en el instante i, tal que i pertenece a T∞.

S_i: asigna un valor de verdad según la siguiente definición:

Función de evaluación.

- $S_i(x)=\xi(x)$, *si x variable TI* .
- $S_i(y)=\eta_i(y)$, *si y variable TD e* $i \in T_f$.
- $S_i(y)=\eta_e(y)$, *si y variable TD e* $i \notin T_f$.
- $S_i(f(t_1,....,t_n))=\Phi(f)(S_i(t_1),.....,S_i(t_n))$, *para toda función n-aria f.*

Dado p una fórmula atómica:

$S_i(p(t_1,.....,t_n))$=*true sii:*

- *p es un predicado TI y* $(S_i(t_1),.....,S_i(t_n)) \in \Pi(p)$ *ó*
- *p es un predicado TD ,* $i \in T_f$ *y* $(S_i(t_1),.....,S_i(t_n)) \in \Pi_i(p)$ *ó*
- *p es un predicado TD ,* $i \notin T_f$ *y* $(S_i(t_1),.....,S_i(t_n)) \in \Pi_e(p)$.

Si A y B son fórmulas cualquiera :

- $S_i (\neg A) = \neg (S_i (A))$
- $S_i (A \wedge B) = S_i (A) \wedge S_i (B)$
- $S_i (\forall(x) (A(x))) = AND_{aj} S_i(A(^x/_{aj}))$, *para todo* $a_j \in D(x)$
- $S_i (Dist (A , t)) = S_{i+v} (A)$ *y* $v=S_i(t)$

Sea S una estructura adecuada para la fórmula TRIO' A. Se define a S como ***modelo*** de A sii ∃ i ∈ T∞ y S_i (A)= True. También A es *válida temporalmente* sii S_i (A) = True para todo i ∈ T∞; es *válida* sii es válida temporalmente en toda estructura adecuada.

Dada una historia, se puede formar una interpretación con los valores tomados por esta para las representaciones de los predicados y variables. Entonces, esta historia es una implementación de un

sistema sii la interpretación obtenida de la historia es un modelo de la fórmula que expresa dicho sistema.

Las variables infinitas cumplen con la principal función para la cual fueron incorporadas, que es la de hacer referencia a expresiones del tipo Always, o sea representar todo el dominio temporal. Las restricciones sobre estas (variables dentro del término temporal del Dist, no anidadas y funciones lineales), permiten obtener un intervalo finito de los valores que deben alcanzar para cumplir su cometido. Debido a que todas las fórmulas sin Dist tienen el mismo valor de verdad (ver capítulo 3) fuera del intervalo finito solo se necesitan los valores de las variables infinitas que evalúan estas fórmulas dentro de la ventana finita y al menos un valor para el resto del dominio temporal, evitando instanciar todos los valores, en este caso infinitos de las variables en cuestión. Esta es la base de la decibilidad de los algoritmos propuestos.

Ejemplo de una especificación en TRIO'

En esta sección se presenta un ejemplo de un sistema descripto con TRIO'. Como primer paso se detalla el lenguaje del sistema, luego la especificación, la que describe el funcionamiento del mismo y por último las propiedades, que son las características deseadas que el sistema debe cumplir.

El sistema representa el funcionamiento de un ascensor, donde se ha omitido hacer referencia a alarmas y al control de puertas por cuestiones de simplicidad. El ascensor se frena 3 unidades de tiempo como mínimo en cada piso y tarda una unidad en moverse de un piso al otro. Recorre en una dirección hasta acabar con todos los pedidos en ese sentido y luego cambia de sentido. Se deben cumplir las siguientes propiedades: una persona no debe esperar más de 150 unidades de tiempo desde que apretó el botón y si no hay pedidos debe estar parado.

El lenguaje del sistema es el siguiente:

(1) Constantes: maxpiso=13, Δt=150.

(2) Variables

- (i) TI: x, y ,t.
- (ii) TD: piso: indica el piso en que está el ascensor.
- (iii) Variable infinita: z.

(3) Dominios:

- (i) D(piso)=D(x)=D(y)=[1..maxpiso].
- (ii) D(t)=[0..Δt].
- (iii) T_f= [0..65535].
- (iv) $T\infty = \mathbb{Z}^+$.
- (v) D(t) = D(z) = $T\infty$.

(4) Predicados TD:

- (i) sube, baja,
- (ii) para: Indica que el ascensor está parado.
- (iii) pedido(x): Indica si el piso x está pedido

Las sentencias que representan el sistema son:

- Control de movimiento:

f_1 : (para $\leftrightarrow$ $\neg$ sube $\wedge$ $\neg$ baja)

f_2 : (sube $\leftrightarrow$ $\neg$ baja)

- Si esta en un piso no esta en otro:

f_3 : $\forall x$ ($\forall y$ (piso=x $\wedge$ $\neg$(x=y)) $\rightarrow$ $\neg$ (piso=y)))

- Si está en el primer piso no puede bajar:

f_4 : (piso=1) $\rightarrow$ Until^P($\neg$ baja,sube)

- Si está en el ultimo piso no puede subir:

f_5 : (piso=maxpiso) $\rightarrow$ Until^P($\neg$ sube,baja)

- Cuando llega, espera por lo menos 3 unidades y hasta que haya otro pedido:

f_6 : Becomes (para) -> (Lasts(para,4) $\wedge$ $\exists$x(Until(para,pedido(x))))

- Si subía y paró, y hay pedido de algún piso superior, entonces sigue subiendo:

f_7 : (para $\wedge$ UpToNow(sube) $\wedge$ $\exists$x (piso<x $\wedge$ pedido(x)))$\rightarrow$ Futr(sube,4)

- Si bajaba y paró, y hay pedido de algún piso inferior, entonces sigue bajando:

f_8 : (para $\wedge$ UpToNow(baja) $\wedge$ $\exists$x (piso>x $\wedge$ pedido(x)))$\rightarrow$ Futr(baja,4)

- Cuando pasa por un piso pedido, para.

f_9 : (sube $\vee$ baja) $\wedge$ pedido(piso) $\wedge$ Uptonow($\neg$ para)) $\rightarrow$ (para $\wedge$ Dist($\neg$ pedido(piso),1))

- Si baja cambia de piso y tarda una unidad de tiempo:

f_{10}: $\forall$(x)(piso=x $\wedge$ baja $\rightarrow$ Dist(piso=x-1,1))

- Si sube cambia de piso y tarda una unidad de tiempo:

f_{11}: $\forall$(x)(piso=x $\wedge$ sube $\rightarrow$ Dist(piso=x+1,1))

Entonces se puede escribir la *Especificación* como la conjunción de todas las sentencias anteriores valiendo en todos los instantes de tiempo, lo que queda expresado en la siguiente fórmula:

$$\Gamma = \text{Always} (\Lambda_i f_i)$$

La especificación anterior deberá cumplir las siguientes condiciones:

- Cada vez que hay un pedido en un piso, en un tiempo menor que Δt se debe llegar al mismo:

  $p_1 : (\forall x \, (\text{pedido}(x) \rightarrow \exists t \, (0 < t < \Delta t \rightarrow \text{Futr}(\text{piso}=x \wedge \text{para}, t))))$

- Si no hay pedidos debe estar parado:

  $p_2 : (\forall x(\neg \text{pedido}(x)) \rightarrow \text{para})$

Por lo que las *Propiedades* se pueden expresar como:

$$\Pi = \text{Always} (\Lambda_i p_i)$$

Entonces se puede probar que la especificación es válida buscando al menos un modelo para Γ. Además si se quiere ver que cumple con las condiciones deseadas, se puede buscar un modelo para la fórmula $\Gamma \wedge \Pi$.

PROPIEDADES

A continuación se presentan un conjunto de propiedades.

Uno de los objetivos de TRIO' es probado en este capítulo: las tautologías y las contradicciones mantienen su valor natural en TRIO', o sea, True y False respectivamente cualquiera sea el instante y la interpretación en que se evalúen , es decir las tautologías son válidas.

Todas las demostraciones están desarrolladas en el apéndice A.

Definición : Si A, B y C son fórmulas cualquiera, entonces los esquemas de axiomas Ax_1, Ax_2 y Ax_3 [Men79]son:

- $Ax_1 : A \rightarrow (B \rightarrow A)$
- $Ax_2 : (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
- $Ax_3 : (\neg A \rightarrow \neg B) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$

Lema 1.1 : Los esquemas de axiomas, Ax_1,Ax_2 y Ax_3, de tal lógica son válidos.

Lema 1.2 : Los axiomas[VIII] de tal lógica son válidos.

Lema 1.3 : La Regla MP conserva la validez lógica. Es decir si A y B son 2 fórmulas cualquiera, A y $A \rightarrow B$ son válidas, entonces B es válida.

Proposición : Con las definiciones anteriores para el $\neg$ y el $\wedge$, todos los teoremas de la lógica proposicional, y por lo tanto todas las tautologías, son válidas en TRIO'.

Corolario : Todas las contradicciones son False para cualquier instante de tiempo de cualquier interpretación.

Proposición : Sea A un teorema de la lógica proposicional, entonces Always(A) es válida.

Proposición : Vale la Temporal Transparency, o sea Dist ($\neg$ A , t) $\equiv \neg$ (Dist (A , t)).

Corolario : Always($\neg$A) $\equiv \neg$ Always (A).

[VIII] Definimos los axiomas en forma usual, o sea si A es un esquema de axioma, Ax1, Ax2 ó Ax3, entonces el reemplazo de las letras de predicados en A, por fórmulas bien formadas es un axioma.

Proposición : Vale Dist (A $\wedge$ B , t) $\equiv$ (Dist (A , t)) $\wedge$ (Dist (B , t)).

Corolario : Vale : Always (A $\wedge$ B) $\equiv$ Always (A) $\wedge$ Always (B).

Corolario : Si valen Always (A) y Always (A $\rightarrow$ B), entonces vale Always (B).

Proposición : Vale : Always (A) $\rightarrow$ Sometimes (A).

Proposición : Si i,j $\notin T_f$ y t es un término cualquiera, entonces $S_i(t)=S_j(t)$.

Proposición : Si i,j $\notin T_f$ y A es una fórmula TRIO' que no contiene el operador Dist entonces $S_i(A)=S_j(A)$.

Corolario : Las siguientes propiedad es válida: $\neg \neg$ A -> A.

Corolario : Si A es una fórmula sin predicados TD y sus términos no contienen variables TD entonces

- ◊ Dist (A, t) $\equiv$ A
- ◊ $S_i(A)=S_j(A)$, $\forall i,j \in T\infty$.

Corolario : No vale Dist (Dist (A , t_1) , t_2) $\equiv$ Dist (A , $t_1 + t_2$).

Con las proposiciones y corolarios demostrados se ha inferido varios importante de resultados, entre los que se pueden distinguir la validez de los teoremas demostrados en el cálculo proposicional y como se dijo anteriormente, la consistencia de las tautologías y las contradicciones con lo que se alcanza uno de los objetivos de TRIO' al tener un valor de verdad natural para tales fórmulas.

ALGORITMO DE GENERACIÓN DE MODELOS

Tomemos una historia del sistema en un instante i, de manera tal que los valores alcanzados por la misma formen una interpretación. Entonces, está interpretación será un estado obtenido por nuestro programa sí y solo sí la fórmula que lo expresa es verdadera para la interpretación en el instante i, es decir es un modelo de la misma. La idea de este algoritmo es buscar al menos un modelo de una fórmula dada (determinar satisfabilidad), para saber si existe una probable evolución del sistema, o sea, si es posible de ser implementada. Además este mismo algoritmo puede utilizarce para generar casos de simulación obteniendo distintos modelos, y por lo tanto distintas historias, de una fórmula dada.

Se cuenta como input con una fórmula bien formada y cerrada TRIO' y con un instante del dominio temporal. Se asume como conocida la parte estática del sistema, lo que se llamará *frame*, o sea, de una temporal structure se conoce D,T,Π, y Φ (ver capítulo 2), tal que Π y Φ asignan valores dentro del dominio que le corresponden, y se buscará la parte dinámica para el instante dado, por lo tanto se obtendrá ηι, ηe, Πe, Πι.

Este algoritmo es decidible siempre que el frame sea parte de una interpretación adecuada y que la longitud de la fórmula sea finita.

Constructivamente primero se divide la fórmula en subfórmulas, hasta llegar a subfórmulas atómicas ground, o sea con todas sus variables instanciadas, creando tantos subárboles como distintas combinaciones haya de variables TD teniendo en cada uno todas las combinaciones necesarias de variables TI.

En cada nodo, además de la fórmula, el instante de tiempo en que se quiere evaluar y los valores tomados por las variables infinitas, se tiene el valor de verdad necesario para satisfacer a la fórmula del nodo raíz. Este último valor evitará asignar valores a los predicados TD que luego deberán ser descartados y además servirá como factor de decisión en el momento de la evaluación.

En la segunda parte se asigna el valor de verdad según el frame dado a la parte estática, y se determina el valor de verdad los predicados TD, teniendo en cuenta los instantes en que se evalúan, generando para cada colisión[IX] 2 subárboles, cada uno con un valor de verdad distinto, de manera de tener todas las combinaciones necesarias de valores para los predicados TD.

[IX] Denominamos colisión a las asignaciones de variables que generan para un mismo predicado y sus términos valores de verdad distintos.

Al finalizar esta etapa, se posee un modelo potencial distinto en cada subárbol hoja.

Para determinar si uno de estos eventuales modelos satisface la fórmula se subirá dentro del subárbol hasta llegar al nodo raíz marcando cada nodo cuyo valor de verdad de la fórmula que contiene no se ha podido alcanzar. El algoritmo encuentra un modelo si al menos el nodo raíz de un árbol está sin marcar.

Algoritmo

Construcción.

Se construirán subárboles cuyos nodos contendrán una 4-upla del tipo (F, i, asg, V), donde F es una fórmula bien formada y cerrada TRIO', V es el valor de verdad que deberá tener la fórmula, $i \in T\infty$, es el instante de tiempo en que se evaluará y $asg \subseteq T\infty$, es un conjunto de valores de las variables infinitas que están libres en la fórmula F o que están en i. Cada subárbol tendrá un conjunto asgTD con una asignación para las variables TD de la forma (y,k,b), $b \in D(y)$ y $k \in T\infty$.

Para mantener un orden entre los distintos subárboles, se organizarán dentro de otra estructura de árbol, que se llamará árbol principal, donde cada nodo tendrá uno de los subárboles antes descriptos. Por lo tanto, ahora se tendrá nodos como los anteriores y nodos que serán subárboles y a menos que se diga lo contrario, cuando se haga referencia a nodos, serán los que contengan las tuplas del tipo (F,i,asg,v) antes descriptas.

Nota: Cada vez que se creen subárboles hijos, se deberán continuar con el algoritmo en los nodos de los subárboles hijos. Esto hará que sean potencialmente válidos solo los subárboles hojas del árbol principal.

Sea F una fórmula cerrada TRIO' :

1. Crear el nodo raíz, del subárbol raíz, con la tupla (F,i,{},T), asgTD={}

2. Repetir hasta que no haya operación posible para todo nodo hoja (F,i,asg,V), de todo subárbol hoja, y F es de la forma:

2.1. $\neg A$

2.1.1 $\neg A$ y V=T : Crear un nodo hijo con (A,i,asg, F).

2.1.2 $\neg A$ y V=F : Crear un nodo hijo con (A,i,asg, T).

2.2. $A \wedge B$: Crear 2 nodos hijos, cada uno con (A,i,asg,V) y (B,i,asg,V) respectivamente.

2.3. ∀xA:

2.3.1 Si x no es una variable infinita: Crear | D(x) | nodos hijos, con ($A^x/_a$, i , asg, V), ∀a ∈ D(x).

2.3.2 Si x es variable infinita: Crear un nodo hijo, con (A , i , asg, V)

2.4. Dist (A , t)

2.4.1. Si t no contiene ninguna variable, entonces:

2.4.1.1 Si |asg| ≤ 1 obtener k como $i+S_i(t)$ y crear un nodo hijo con (A , k ,asg , V).[X]

2.4.1.2 En caso contrario, si x es la variable infinita que está en i obtener:

temp = { x'/ $S_i(t)+i^x/_{x'} \in T_f$, x' ∈ asg }

temp'= asg - temp

Si temp ≠ {} : Crear | temp | nodos hijos, con (A, $S_i(t)+i^x/_{x'}$,{x'} ,V) , ∀x' ∈ temp.

Crear un nodo hijo con (A, $S_i(t)+i$,temp' ,V).

2.4.2. Si t contiene solo variables TI y valores ground[XI]*, entonces si x es la variable infinita, hacer:.*

temp = { x'/ $i+t^x/_{x'} \in T_f$, x' ∈ D(x) }

temp'= D(x) - temp

Si temp ≠ {} : Crear | temp | nodos hijos , con (A, $i+S_i(t^x/_{x'})$,{x'} ,V) , ∀x' ∈ temp.

Crear un nodo hijo con (A, i+t ,temp' ,V).

2.4.3. Si t contiene una variable TD z , entonces

2.4.3.1 Si i es ground e i ∈ T_f

2.4.3.1.1 Si (z , i , b) ∈ asgTD, b ∈ D(z), entonces crear un solo nodo hijo con (Dist(A, $t^z/_b$), i , asg, V).

2.4.3.1.2. En caso contrario, crear | D(z) | subárboles hijos, con el mismo subárbol que el padre, pero cada uno con un nodo con (Dist(A, $t^z/_a$) ,i ,asg ,V) hijo del nodo (Dist(A, t) ,i ,asg ,V), para cada a ∈ D(z) y con asgTD = asgTD ∪ { (z, i,a) }.

2.4.3.2 Si i no es ground o i ∉ T_f

2.4.3.2.1 Si (z , j , b) ∈ asgTD, j ∉ T_f, b ∈ D(z), entonces crear un solo nodo hijo con (Dist(A, $t^z/_b$), i , asg, V).

[X] Si i es una expresión obtener $i=i^x/_a$ tal que a ∈ asg y x es la variable infinita en i.

[XI] Observar que este caso se da solo para la variable infinita, ya que por ser las fórmulas cerradas todas las variables del término t ya han sido instanciadas a esta altura. Además i es un valor ground por no haber variables infinitas anidadas.

2.4.3.2.2. En caso contrario, crear | D(z) | subárboles hijos, con el mismo subárbol que el padre, pero cada uno con un nodo con (Dist(A, $t^z/_a$) ,i ,asg ,V) hijo del nodo (Dist(A, t) , i ,asg ,V), para cada a ∈ D(z) y con asgTD = asgTD ∪{ (z, k,a) }, k ∈ T∞ - T_f.

3. Para todo nodo (F,i,asg,V) que es una hoja de un subárbol hoja, F fórmula atómica y F tiene variables TD sin instanciar entonces para cada variable TD z, hacer:

3.1 Si i no es ground, entonces si x es la variable que está en i, obtener $i=i^x/_b$, b ∈ asg.

3.2 Si i ∈ T_f, entonces

3.2.1 Si (z , i , b) ∈ asgTD, b ∈ D(z), crear un solo nodo hijo con (($F^z/_b$), i ,asg , V).

3.2.2 En caso contrario crear | D(z) | subárboles hijos, con el mismo subárbol que el padre, pero cada uno con un nodo con (($F^z/_a$), i , asg, V) hijo del nodo (F, i , asg, V), para cada a ∈ D(z) y con asgTD = asgTD ∪{ (z, i,a) }.

3.3 En caso contrario,

3.3.1 Si (z , j , b) ∈ asgTD, j ∉ T_f, b ∈ D(z), crear un solo nodo hijo con (($F^z/_b$), i ,asg , V).

3.3.2 En caso contrario crear | D(z) | subárboles hijos, con el mismo subárbol que el padre, pero cada uno con un nodo con (($F^z/_a$), i , asg, V) hijo del nodo (F, i , asg, V), para cada a ∈ D(z) y con asgTD = asgTD ∪{ (z, i,a) }.

Evaluación

Esta parte del algoritmo intenta buscar un modelo para la fórmula, lo que dirá si es o no satisfacible. Para esto se utilizan los subárboles que son hojas del árbol principal. Se agrega a cada subárbol los subconjuntos $\Pi_i'(p)$, $\Pi_i(p)$, $\Pi e(p)$ ó $\Pi_e'(p)$, i ∈ T_f para cada predicado TD p que serán asignados según sea el instante de tiempo, donde cada uno está definido como Π_i en la interpretación y se inicializarán como el conjunto vacío. Los conjuntos $\Pi_i'(p)$ y $\Pi_e'(p)$ serán análogos a los definidos en la estructura y representan las asignaciones a los predicados TD cuyo valor de verdad es False.

La idea principal es marcar cada nodo del subárbol si no se puede alcanzar el valor de verdad indicado en el mismo, hasta llegar al nodo raíz.

i) Para todo nodo hoja de cada subárbol hoja :

Dado una hoja (F,i,asg,V) , F es una fórmula atómica $p(v_1,.....,v_n)$:

1 Si p es un predicado TI

1.1. Si V=F y la tupla de valores $(v_1,.....,v_n) \in \Pi(p)$ *, entonces marcar dicho nodo.*

1.1. Si V=T y la tupla de valores $(v_1,.....,v_n) \notin \Pi(p)$ *, entonces marcar dicho nodo.*

2 Si p es un predicado TD [XII]*:*

2.1 Si i $\in T_f$

** Si V=T*

2.1.1 Si $(v_1,.....,v_n) \in \Pi'_i(p)$*, entonces :*

Crear dos subárboles hijos, iguales al padre.

En un subárbol hijo hacer :

$\Pi_i(p) = \Pi_i(p) \cup \{(v_1,.....,v_n)\}$

$\Pi'_i(p) = \Pi'_i(p) - \{(v_1,.....,v_n)\}$

marcar todos los nodos tal que hicieron la asignación de $(v_1,.....,v_n)$ *a* $\Pi'_i(p)$ *,*

En el otro subárbol hijo marcar el correspondiente al nodo actual.

2.1.2 En caso contrario, hacer $\Pi_i(p) = \Pi_i(p) \cup \{(v_1,.....,v_n)\}$.

** Si V=F*

2.1.3 Si $(v_1,.....,v_n) \in \Pi_i(p)$*, entonces copiar el árbol,*

Crear dos subárboles hijos, iguales al padre.

En un subárbol hijo hacer :

$\Pi'_i(p) = \Pi'_i(p) \cup \{(v_1,.....,v_n)\}$

$\Pi_i(p) = \Pi_i(p) - \{(v_1,.....,v_n)\}$

marcar todos los nodos tal que hicieron la asignación de $(v_1,.....,v_n)$ *a* $\Pi_i(p)$ *,*

En el otro subárbol hijo marcar el correspondiente al nodo actual.

2.1.4 En caso contrario, hacer : $\Pi'_i(p) = \Pi'_i(p) \cup \{(v_1,.....,v_n)\}$

2.2 Si i $\notin T_f$

Análogamente con $\Pi e(p)$ *y* $\Pi' e(p)$ *,*

ii) Para todo nodo(F,i,asg,V) tal que ya se trato a todos sus hijos:

1 Si F es una fórmula atómica, F tiene variable TD sin instanciar:

1.1 Si su hijo está marcado, entonces marcar el nodo.

[XII] Ídem nota X

1.2 En caso contrario, copiar asg.

2. Si F=A ∧ B:

2.1 Si V=T

Si alguno de sus hijos esta marcado, marcar este nodo.

En caso contrario, obtener asg como la intersección de los asg de los nodos hijos tal que las fórmulas contienen una variable infinita libre o que contienen la variable infinita en i.

2.2 Si V=F

Si sus dos hijos están marcados, marcar este nodo.

En caso contrario, obtener asg como la unión de los asg de los nodos hijos tal que las fórmulas contienen una variable infinita libre o que contienen la variable infinita en i y que no están marcados.

3. Si F= ¬A:

Si el nodo hijo esta marcado, entonces marcar el nodo.

En caso contrario, copiar asg.

4. Si F= ∀x A :

4.1 Si x no es la variable infinita

4.1.1 marcar el nodo si:

V=T y alguno de sus hijos está marcado,

ó , V=F y todos sus hijos están marcados.

4.1.2 en caso contrario si la fórmula F tiene una variable infinita libre o la variable infinita está en i, entonces si :

V=T obtener asg como la intersección de los asg de los nodos hijos

V=F obtener asg como la unión de los asg de los nodos hijos que no están marcados.

4.2 Si x es la variable infinita marcar el nodo si:

su hijo está marcado,

ó , V=T y asg de su hijo es distinto de D(x) ,

ó , V=F y asg de su hijo es igual al conjunto vacío.

5. Si F= Dist(A,t):

Si todos sus hijos están marcados, entonces marcar el nodo,

En caso contrario, obtener asg como la unión de los asg de todos sus hijos sin marcar

iii) El algoritmo termina:

Si se llego al nodo raíz de alguno de alguno de los subárboles hojas y está sin marcar

o se llegó al nodo raíz de todos los subárboles hojas.

El algoritmo es ***exitoso***, si terminó por la primer opción.

Observaciones:

En el paso 2.3.1 de la construcción de los subárboles se asignan todos los valores de las variables de las variables TI. Igualmente en los pasos 2.4.3 y 3 para los valores de las variables TD en forma consistente.

En el paso 2.4.1 se pregunta por la longitud de asg ya que en el único caso que puede ser mayor que 1 es cuando i representa una expresión donde está la variable infinita y asg indica más de un valor para la misma, (se está fuera de T_f), por lo que hay que volver a calcular el valor de tal variable por si el nuevo término pertenece a T_f.

Notar que cuando se construye Temp, dado que se pide funciones lineales para los términos con variables infinitas, se puede determinar buscando los valores que hacen al término F(x) estar en el domino T_f, o sea hacer $F(x) \geq T_{min}$ y $F(x) \leq T_{max}$, siendo T_{min} y T_{max} los valores máximos y mínimos de T_f, por lo que quedan 2 ecuaciones lineales con una incógnita, cuyo resultado es un intervalo finito. Notar, además, que no se podría asegurar que temp sea finito si tuviera más de una variable infinita anidada (lo que dejaría F(x) con más de una incógnita) o si tuviera alguna función no lineal (podría volver a caer en el intervalo infinitas veces). Ejemplo: T_f=[1..10], x e y son variables infinitas e F(x)=3-y+x o F(x)=10+sen(x).

Temp' se deberá expresar por comprensión, ya que siempre es infinito y distinto de vacío, por ser la diferencia entre un conjunto infinito y uno finito.

En el paso i.2 de la parte de evaluación si el subárbol ya tiene una asignación para un predicado TD en el mismo instante de tiempo y para los mismos valores de sus términos, pero con distinto valor de verdad, entonces, se generan dos árboles, cada uno con un valor de verdad distinto para tal predicado para mantener la consistencia y todas las posibles combinaciones.

En ii.2.1 y ii.4.1.2 de la misma parte se podrían eliminar las asignaciones a predicados TD (Π_t, Π'_t, etc.) de las asignaciones eliminadas de la variable infinita, pero no sería obligatorio debido a

que ya no va a haber nuevas asignaciones a los predicados, por lo que estos valores no afectarán el valor de verdad de la fórmula raíz, ya que su eliminación solo indicaría que los mismos pueden alcanzar cualquier valor.

Notar que se crean subárboles solo en los casos donde hay nuevas asignaciones para las variables y predicados TD. Estos subárboles son siempre iguales a su padre, excepto para los nodos con la variable o el predicado que los generó.

Este mismo algoritmo puede ser utilizado, como dijimos anteriormente, para generar distintos modelos de una misma fórmula cambiando la condición de terminación y tomando como interpretaciones modelos a todas las obtenidas de los subárboles hojas sin marcar. Cada modelo obtenido puede utilizarce como un caso de simulación de una eventual implementación.

Construcción de la interpretación.

Una vez terminado el algoritmo en forma exitosa, podremos construir una interpretación según el frame dado y el resultado obtenido de los conjuntos asgTD, Π_ι, Π'_ι, etc., del subárbol cuya raíz está sin marcar, de la siguiente manera:

Para toda (z , i , a) $\in$ asgTD hacer:

$$* \text{ Si } i \in T_f : \eta\iota (z) = a$$

$$* \text{ Si } i \notin T_f : \eta e (z) = a$$

Copiar $\Pi\iota$ y Πe para todo predicado TD p, tal que fue asignado en el algoritmo.

Notar que a partir de un resultado conseguido por algoritmo se podrá construir más de una interpretación que sea modelo, dado que los valores no asignados a los predicados TD en ninguno de los conjuntos Π_ι, Π'_ι, Π_e, Π'_e, etc., para ciertos instantes de tiempo, indica que las asignaciones a dicho predicado en tal instante no modifican la satisfabilidad de la fórmula, por lo que el mismo puede alcanzar cualquier valor. Entonces la interpretación anteriormente construida es una de las varias posibles.

Ejemplo

Continuando con el ejemplo del ascensor, se buscará un modelo para una de las fórmulas que están en la especificación. Se le agregará una cuantificación sobre el dominio temporal para que valga en todo instante. Esto lo haremos utilizando la variable infinita y evaluando en algún valor del dominio temporal, en este caso 3.

Para este ejemplo se utilizará el siguiente frame, en el que se modificará algunos valores del lenguaje del ejemplo por razones de simplicidad en algunos casos :

- Constantes: maxpiso=2.
- Funciones +,= : Son asumidas en forma usual.
- Variables TI: x.
- TD: piso.
- Variable infinita: z.
- Dominios: D(x) = D(piso) = [1..maxpiso], D(z) = $\mathbb{Z}^+$.
- T_f=[2..4] , T∞ = $\mathbb{Z}^+$.
- Predicados TD:sube.

Se Quiere evaluar f_{11}, o sea : Si sube, cambia de piso y tarda una unidad de tiempo, es decir:

$\forall(x)(piso=x \wedge sube \rightarrow Dist(piso=x+1,1))$

Para que valga siempre, se le agrega Always, por lo tanto la fórmula para evaluar es:

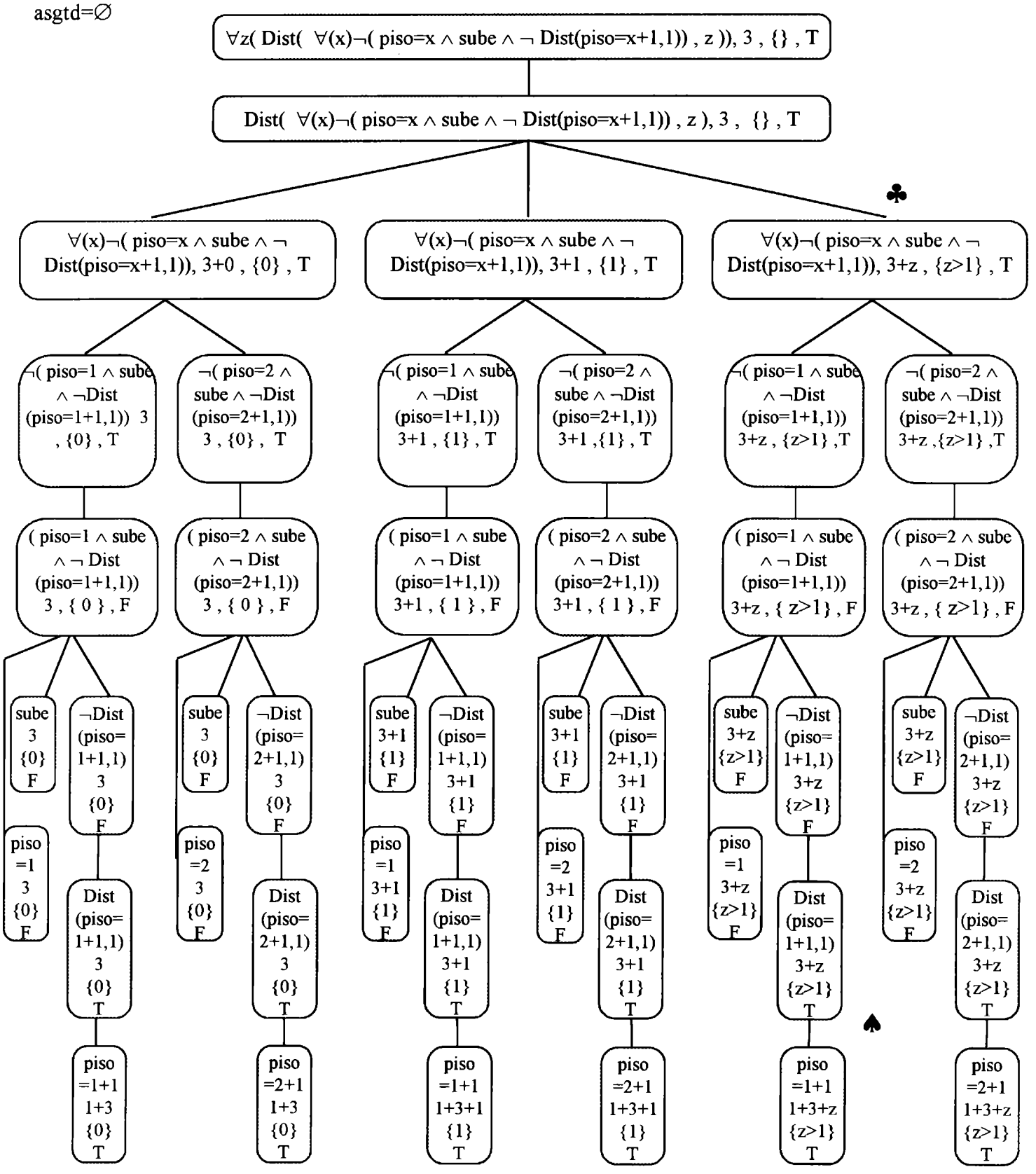
$Always\ (\ \forall(x)(piso=x \wedge sube \rightarrow Dist(piso=x+1,1))\ )$

lo que queda expresado en el lenguaje TRIO' como:

$\forall z(\ Dist(\ \forall(x)\neg(\ piso=x \wedge sube \wedge \neg\ Dist(piso=x+1,1)),\ z\ )\ )$

A continuación se desarrolla el subárbol raiz :

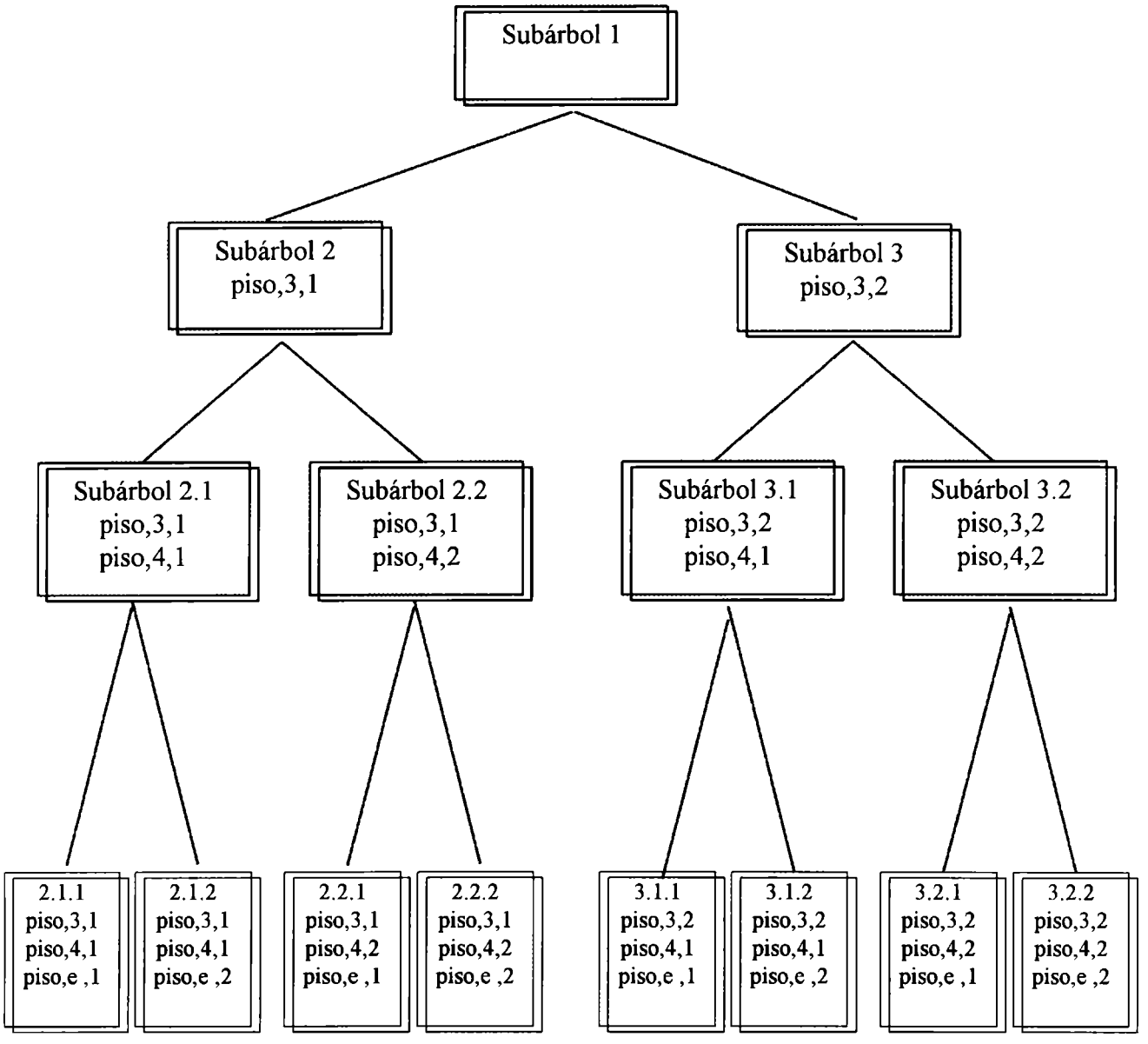
Subárbol 1



♣ Temp = {z' | $2 \le 3+z' \le 4$, $z' \ge 0$}={ 0,1} , $\Rightarrow$ Temp' = {z > 1}

♠ Temp = {z' | $2 \le 3+z' \le 4$, $z' > 1$}={} , $\Rightarrow$ Temp' = {z > 1}

Debido a que en los nodos hojas hay variables TD, es que se hace necesario la generación de subárboles según el siguiente diagrama:

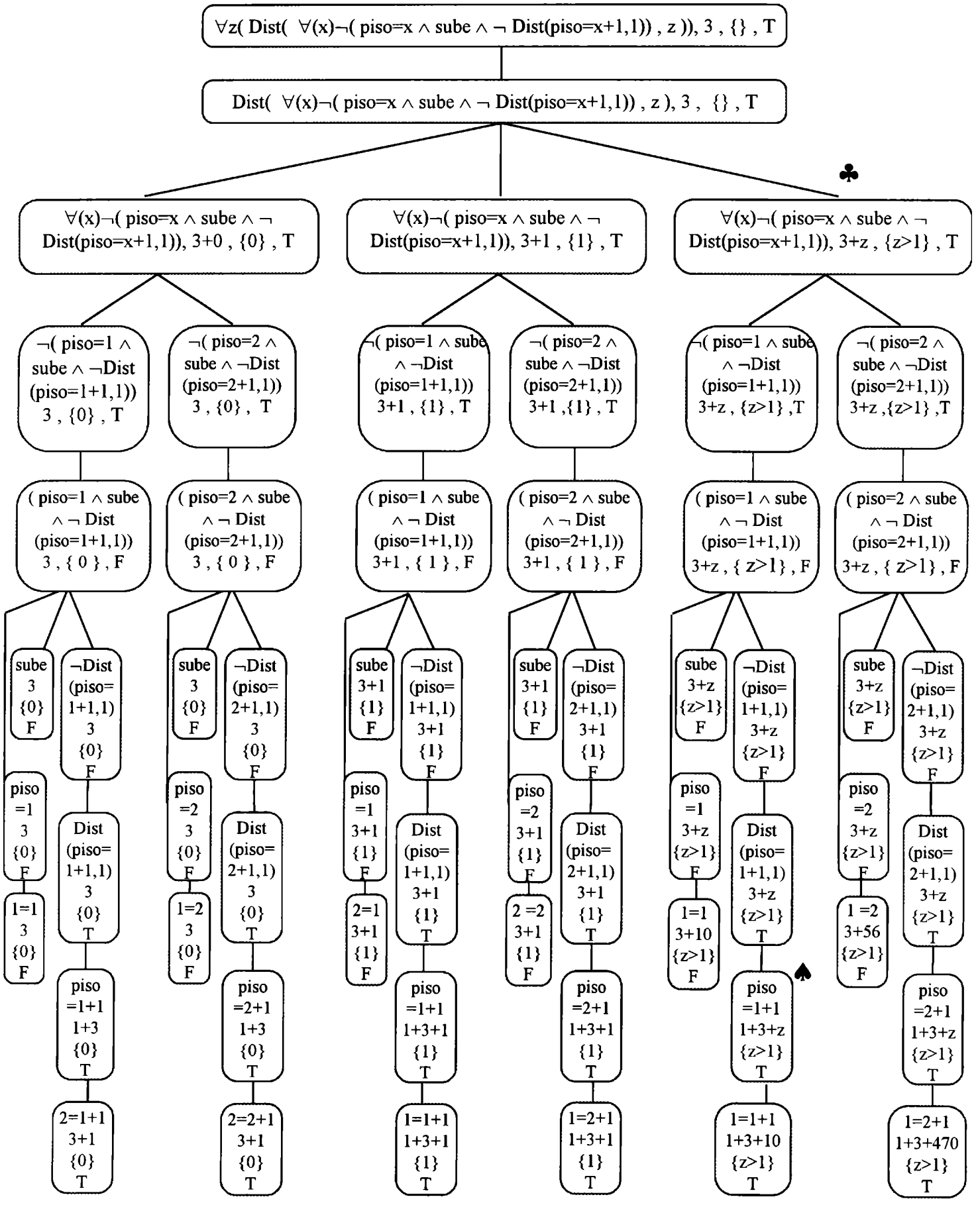


Notación: e indica cualquier instante de tiempo fuera del intervalo finito T_f.

Para no detallar todos los subárboles mostraremos los valores de las variables TD que hacen crear nuevos árboles y tomaremos uno de ellos como ejemplo.

Subárbol 2.2.1

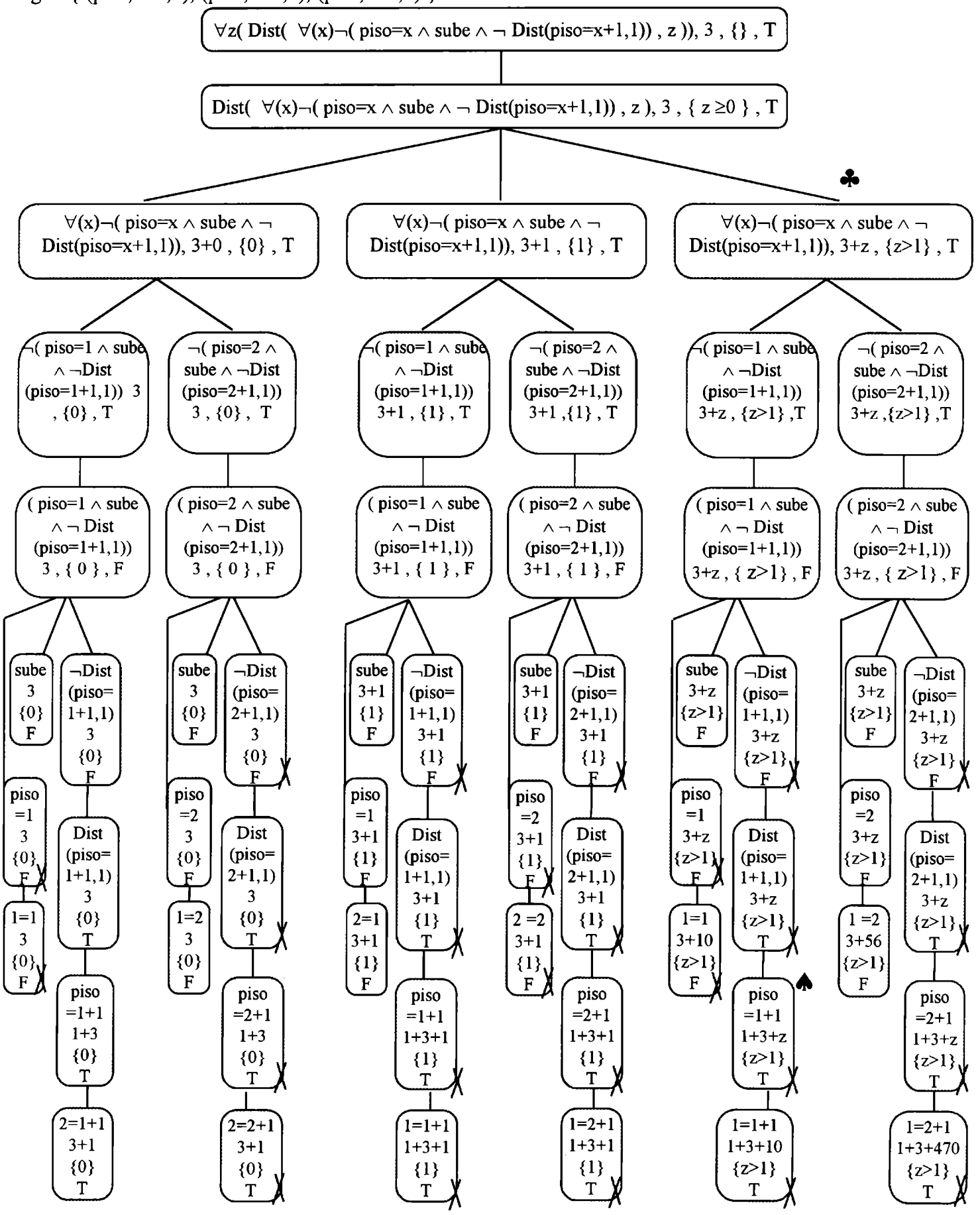
asgtd={ (piso,3+0,1), (piso,3+1,2), (piso,3+2,1) }



A continuación vamos a evaluar este Subárbol.

Subárbol 2.2.1

asgtd={ (piso,3+0,1), (piso,3+1,2), (piso,3+2,1) }



Los valores obtenidos para los predicados son: $\Pi'_3(sube)=\{\varnothing\}$, $\Pi'_4(sube)=\{\varnothing\}$ y $\Pi'_e(sube)=\{\varnothing\}$

Por cuestiones de simplicidad se ha desarrollado un solo subárbol, que por haber llegado al nodo raíz sin marcar y por lo tanto ser exitoso lo hace suficiente para buscar la satisfabilidad.

Se deberá notar que se ha simplificado un paso en donde existe la disyunción de tres fórmulas, obteniendo los tres hijos con un término cada uno, en vez de asociar dos términos y obtener dos hijos, para luego procesar al termino que contiene la asociación. Es claro que este proceder no implica cambio en la evaluación.

Una de las posibles interpretaciones derivadas a partir del resultado del algoritmo conforme a lo indicado en el subárbol elegido por los conjuntos asgTD, Π_i , Π'_i , Π_e , Π'_e ,etc. es:

- Constantes: maxpiso=2.
- Funciones +,= : Son asumidas en forma usual.
- Variable TD: piso: .
- Variable infinita: z .
- Variables TI: x.
- Dominios: D(x) = D(piso) = [1..maxpiso], D(z) = $\mathbb{Z}^+$.
- $T_f=[2..4]$, $T\infty = \mathbb{Z}^+$.
- η_3 (piso) = 1 , η_4 (piso) = 2 , η_e (piso) = 1
- Predicados TD sube : Π_2(sube)={}, Π_3(sube)={}, Π_4(sube)={}, Π_e(sube)={}.

Demostración para el algoritmo de Generación de Modelos

La idea de la demostración se basa en probar el siguiente enunciado:

Sea A una fórmula bien formada y cerrada cualquiera de TRIO'. Entonces existe una interpretación (temporal structure), construida con los valores dados por el algoritmo (asgTD, Π_i, Π'_e, etc.) y con el frame dado tal que :

1) satisface A en el instante i (S_i (A) = true) sii el algoritmo comenzando con (A,i,T,{}) es exitoso,

2) no satisface A en el instante i (S_i (A) = false) sii el algoritmo comenzando con (A,i,F,{}) es exitoso,

Para esto se hace inducción sobre la estructura de una fórmula TRIO' y en el caso base particularmente, se hace otra inducción, pero sobre la cantidad de variables TD.

Como se dijo anteriormente la demostración completa está desarrollada en el apéndice B.

Complejidad

La complejidad del algoritmo es, como se podía intuir, exponencial cuya base son los dominios y su exponente es la longitud de la fórmula.

Para realizar su cálculo se ha tomado la construcción de cada nodo como un solo paso, ya que su tiempo no afecta el tiempo total.

Las iteraciones que generan mayor cantidad de nodos hijos son aquellas en las que se instancian los valores para las variables TI, por lo que se obtienen para cada variable un nodo hijo por cada elemento de su dominio y en el caso particular de las variables infinitas se crean a lo sumo el cardinal de la ventana temporal finita más uno. Por lo que el número máximo de hijos nodos será el mayor entre todos los cardinales de los dominios de tales variables (no infinitas), y el cardinal de T_f mas 1. A este número se lo denominará **mti**, entonces en el nivel k de este árbol habrá a lo sumo mti^k nodos.

Si una fórmula tiene longitud **n**, entonces la profundidad máxima de una rama dentro de un árbol es a lo sumo n, ya que cada vez que se construye un nodo hijo se reduce la fórmula del nodo padre, por lo que todo árbol de nodos tiene a lo sumo n niveles, con lo que se puede inferir que el orden de dicho árbol es **O(mti^n)**

Posteriormente la cantidad de árboles hijos que cada árbol puede generar depende de sus variables TD, por lo que la máxima cantidad de dichos hijos es a menor o igual que el mayor cardinal de los dominios de las variables TD, al que se llamará **mtd**. De la misma forma que para un árbol particular la máxima longitud de una rama de árboles es la longitud de la fórmula, por lo que la cantidad de árboles hojas es menor que **mtd^n**.

Para cada uno de estos mtd^n árboles hojas, con a lo sumo mti^n hojas, existirán como máximo $\binom{mti^n}{2}$ = **mti^n (mti^n-1)/2** colisiones, generando para cada una dos árboles nodos hijos, por lo que se tendrán a lo sumo mtd^n . $2(mti^n (mti^n-1)/2) = mtd^n . mti^n (mti^n-1) =$ **$mtd^n . (mti^{2n} - mti^n)$** árboles hijos, cada uno con mti^n nodos hojas, entonces orden total se puede escribir como $mtd^n . (mti^{2n} - mti^n) . mti^n$, o sea **$O(mtd^n . mti^{3n})$.**

ALGORITMO DE HISTORY-CHECKING

Se conoce una especificación y sus propiedades descriptos como una fórmula TRIO' y un conjunto de valores del sistema que representan su *historia.* Si esta historia se expresa como una interpretación (ver capítulo 2), el objetivo es saber si la misma es un posible estado alcanzado por la especificación y sus propiedades en un instante determinado, o sea si esta interpretación satisface la fórmula.

Entonces, dada una fórmula bien formada y cerrada TRIO' de longitud finita, y una temporal structure, se determinará si dicha fórmula se satisface en el instante j del dominio temporal. O sea su valor de verdad es True, para la interpretación dada, en el instante indicado. Se supone que esta temporal structure es adecuada para la fórmula.

Notar que se cuenta con D, T, Π, Φ, ηι, ηe, Πe, Πι de la interpretación

Este problema es muy parecido al de generación de modelos, pero en este caso se cuenta con los valores de las variables y predicados TD. Por lo tanto, no es necesaria la construcción de los subárboles para buscar dichos valores. Por lo que hace a este algoritmo similar a la generación de un subárbol particular del algoritmo anterior.

Constructivamente primero se divide la fórmula en subfórmulas, hasta llegar a subfórmulas atómicas ground, o sea con todas sus variables instanciadas. En cada nodo se cuenta con la fórmula, el instante de tiempo en que se quiere evaluar, un conjunto que indica los valores que toman las variables infinitas que están libres en la fórmula o las que están en el instante de tiempo y el valor de verdad que se necesita para satisfacer a la fórmula del nodo raíz. Luego se chequeará el valor de verdad según la interpretación dada, para terminar subiendo en el árbol hasta llegar al nodo raíz. Cada nodo marcado indica que no se pudo alcanzar el valor de verdad esperado para el mismo. El valor de verdad se utilizará para determinar la marca en el caso de una fórmula con un cuantificador para la variable infinita.

Algoritmo

Construcción del árbol

En este árbol los nodos contendrán una 4-upla del tipo (F, i, asg, V), donde F es una fórmula cerrada TRIO', V es el valor de verdad que deberá tener la fórmula, i $\in$ T∞ es el instante de tiempo en que se evaluará y asg $\subseteq$ T∞, es un conjunto que contiene las asignaciones a las variables infinitas que están en i o libres en F.

1. Crear en el nodo raíz la tupla (F,i,{},T).

2. Repetir hasta que no halla operación posible para toda hoja del árbol tal que (F,i,asg,V) esta en ella y F es de la forma:

2.1. $\neg A$

2.1.1 $\neg A$ y V=T : Crear un nodo hijo con (A, i,asg, F).

2.1.2 $\neg A$ y V=F : Crear un nodo hijo con (A, i,asg, T).

2.2. $A \wedge B$: Crear 2 nodos hijos, cada uno con (A, i,asg, V) y (B, i, asg, V) respectivamente.

2.3. $\forall x A$:

2.3.1 Si x no es variable infinita, entonces, crear | D(x) | nodos hijos, cada uno con ($A^x/_a$, i , asg, V), $\forall a \in D(x)$.

2.3.2 Si x es variable infinita, entonces, crear un nodo hijo con (A, i, asg, V).

2.4. Dist (A , t)

2.4.1. Si t no contiene ninguna variable TI, entonces:

2.4.1.1 Si |asg| $\leq$ 1 obtener k como $i+S_i(t)$ y crear un nodo hijo con (A , k ,asg , V).[XIII]

2.4.1.2 En caso contrario, si x es la variable infinita que está en i obtener:

temp = { x' / $S_i(t)+i^x/_{x'} \in Tf$, $x' \in asg$ }

temp' = asg - temp

Si temp $\neq$ {} : Crear | temp | nodos hijos, con (A, $S_i(t)+i^x/_{x'}$,{x'} ,V), $\forall x' \in temp$.

Crear un nodo hijo con (A, $S_i(t)+i$,temp' ,V).

2.4.2. Si t contiene solo variables TI y valores ground[XIV]*, entonces si x es la variable infinita, hacer:.*

temp = { x' / $i+t^x/_{x'} \in Tf$, $x' \in D(x)$ }

temp' = D(x) - temp

Si temp $\neq$ {} : Crear | temp | nodos hijos, con (A, $i+S_i(t^x/_{x'})$,{x'} ,V), $\forall x' \in temp$.

[XIII] Si i es una expresión obtener $i=i^x/_a$ tal que a $\in$ asg.

[XIV] Observar que este caso se da solo para la variable infinita, ya que por ser las fórmulas cerradas todas las variables del término t ya han sido instanciadas a esta altura.

Crear un nodo hijo con (A, i+t ,temp' ,V).

3. Para todo nodo (F,i, asg,V) que es una hoja del árbol, F fórmula atómica e i tiene una variable z sin instanciar, reemplazar z en i por a, a ∈ asg.

4. Para todo nodo (F,i, asg,V) que es una hoja del árbol, F fórmula atómica y F tiene variables TD sin instanciar, reemplazar cada variable TD z en F por $\eta_i(z)$.

Evaluación

Esta parte del algoritmo intenta buscar el valor de verdad de la fórmula, para esto utilizaremos el árbol anteriormente construido.

i) Comenzar por las hojas y recorrer el árbol de abajo hacia arriba.

Dado una hoja (F,i,asg,V), F es una fórmula atómica $p(v_1,.....,v_n)$[xv]*:*

1 Si p es un predicado TI

1.1. Si V=F y la tupla de valores $(v_1,.....,v_n) \in \Pi(p)$, entonces marcar dicho nodo.

1.2. Si V=T y la tupla de valores $(v_1,.....,v_n) \notin \Pi(p)$, entonces marcar dicho nodo.

2 Si p es un predicado TD :

2.1 Si $i \in T_f$

2.1.1. Si V=F y la tupla de valores $(v_1,.....,v_n) \in \Pi_i(p)$, entonces marcar dicho nodo.

2.1.2. Si V=T y la tupla de valores $(v_1,.....,v_n) \notin \Pi_i(p)$, entonces marcar dicho nodo.

2.2 Si $i \notin T_f$

2.2.1. Si V=F y la tupla de valores $(v_1,.....,v_n) \in \Pi_e(p)$, entonces marcar dicho nodo.

2.2.2. Si V=T y la tupla de valores $(v_1,.....,v_n) \notin \Pi_e(p)$, entonces marcar dicho nodo.

ii) Para todo nodo(F,i,asg,V) tal que ya se trato a todos sus hijos:

1. Si $F=A \wedge B$:

1.1 marcar este nodo si:

V=T y alguno de sus hijos está marcado,

ó , V=F y sus dos hijos están marcados.

[xv] Notar que por ser hoja de un subárbol, F e y es ground.

1.2 en caso contrario si:

V=T obtener asg como la intersección de los asg de los nodos hijos tal que las fórmulas contienen una variable infinita libre o que contienen la variable infinita en i,

V=F obtener asg como la unión de los asg de los nodos hijos tal que las fórmulas contienen una variable infinita libre o que contienen la variable infinita en i y que no están marcados.

2. Si F= ¬A:

2.1 Si el nodo hijo está marcado, entonces marcar el nodo.

2.2 en caso contrario, copiar asg.

3. Si F= ∀x A :

3.1 Si x no es la variable infinita

3.1.1 marcar el nodo si:

V=T y alguno de sus hijos está marcado,

ó , V=F y todos sus hijos están marcados.

3.1.2 en caso contrario, si i contiene una variable infinita o la F tiene una variable infinita libre entonces si :

V=T obtener asg como la intersección de los asg de los nodos hijos

V=F obtener asg como la unión de los asg de los nodos hijos que no están marcados.

3.2 Si x es la variable infinita, marcar el nodo si:

su hijo está marcado,

ó , V=T y asg de su hijo es distinto de D(x),

ó , V=F y asg de su hijo es igual al conjunto vacío.

4. Si F= Dist(A,t).

4.1 Si todos sus hijos están marcados, entonces marcar el nodo.

4.2 En caso contrario obtener asg igual a la unión de los asg de todos sus hijos sin marcar.

iii) El algoritmo termina cuando se llegó al nodo raíz.

iv) El algoritmo es ***exitoso****, si el nodo raíz está sin marcar.*

Si el algoritmo es exitoso diremos que la interpretación es un modelo de la fórmula dada.

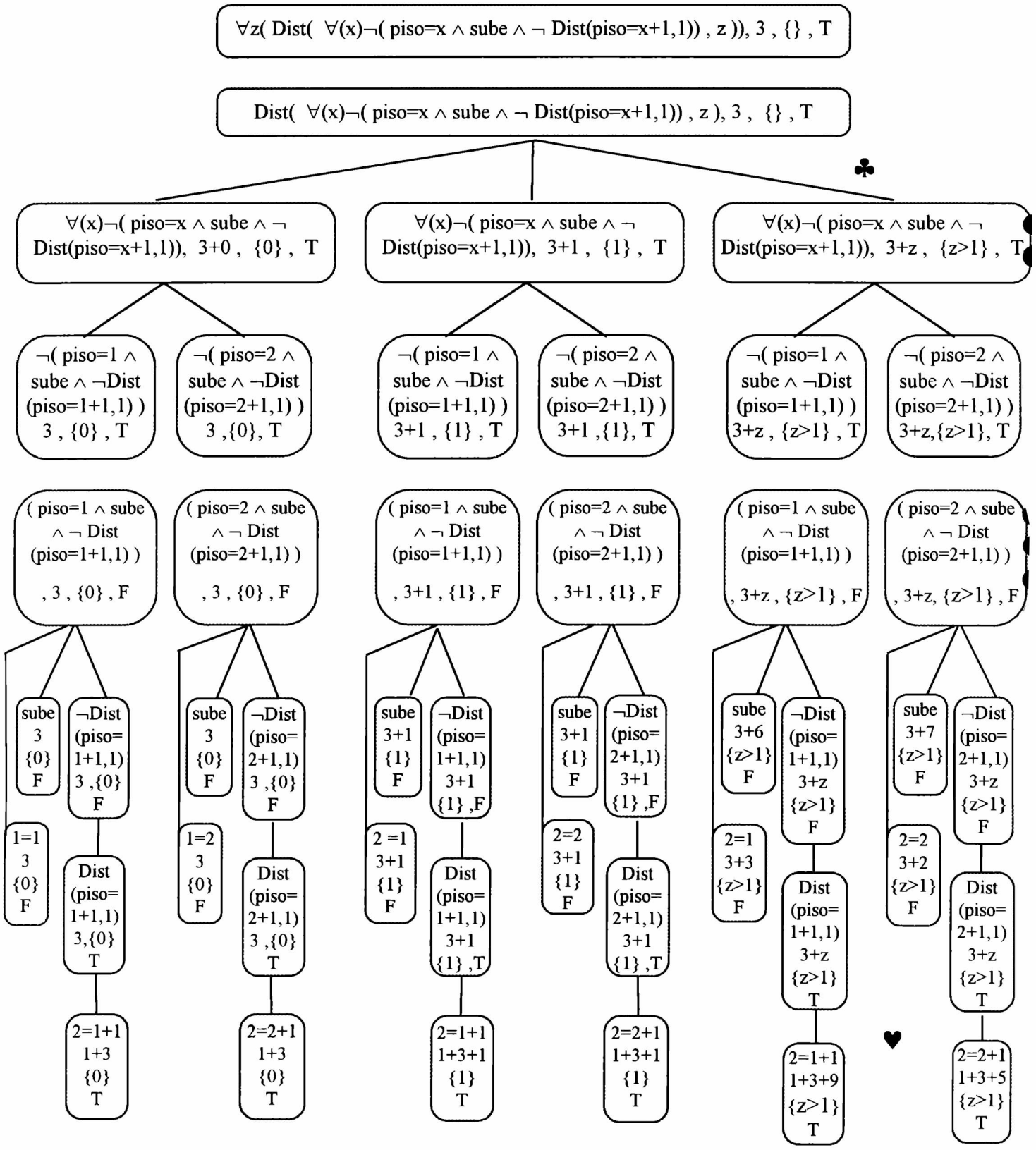
Ejemplo

Volviendo con el ejemplo del ascensor, se evaluará la misma fórmula que se utilizó para el algoritmo de generación de modelos contra la historia representada con la siguiente interpretación:

- Constantes: maxpiso=2.
- Funciones +,= : Son asumidas en forma usual.
- Variables TI: x
- TD: piso: η_3 (piso) = 1 , η_4 (piso) = 2 , η_e (piso) = 2
- Variable infinita: z , D(z) = $\mathbf{Z}^+$.
- Dominios: D(x)=D(piso)=[1..maxpiso]
- T_f=[2..4] , T∞ = $\mathbf{Z}^+$.
- Predicados TD sube : Π_2(sube)={( )} , Π_3(sube)={( )} , Π_4(sube)={} , Π_e(sube)={}.

Como se ha dicho anteriormente, la fórmula TRIO' es:

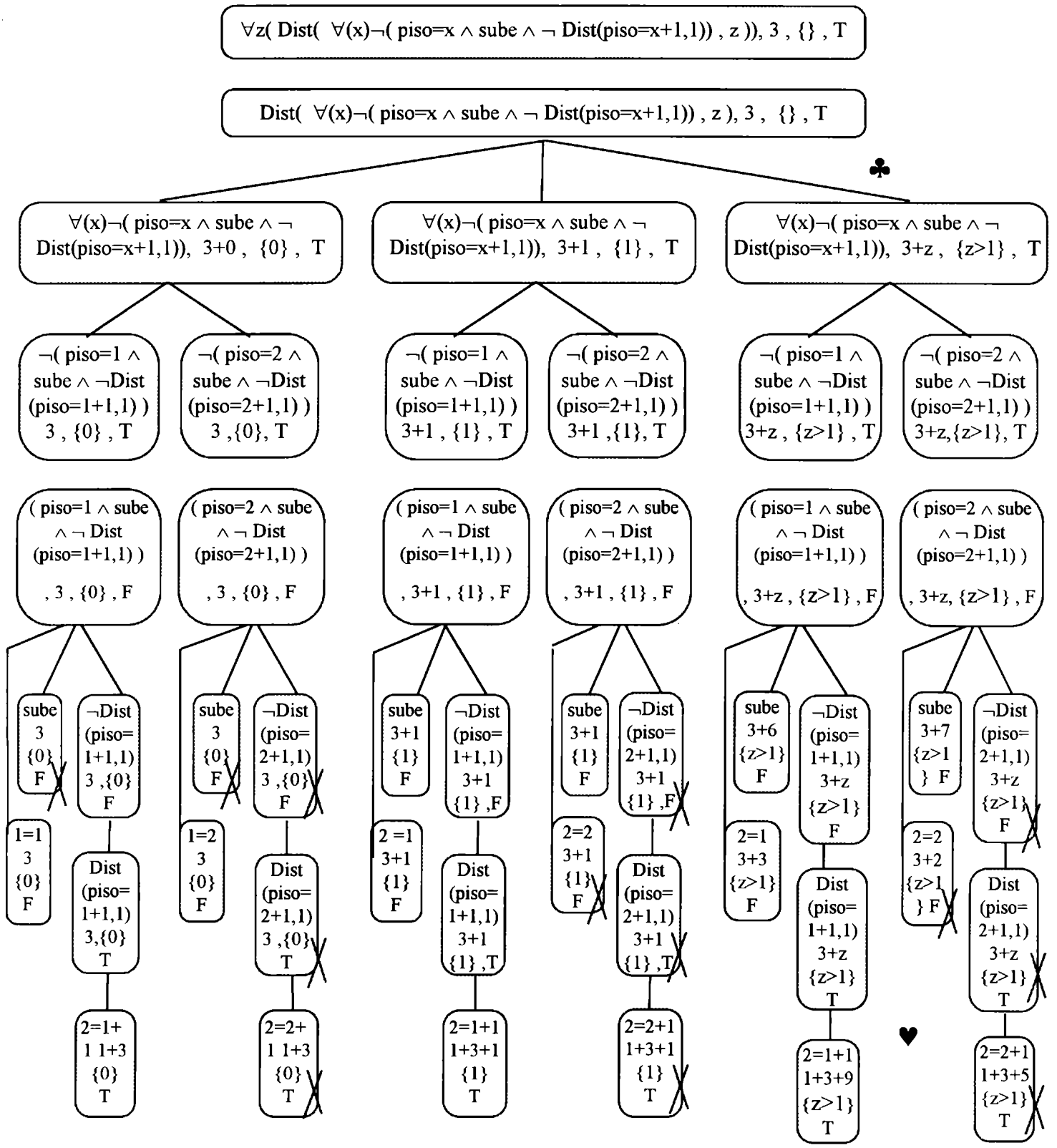
∀z(Dist(∀(x)¬(piso=x ∧ sube ∧ ¬ Dist(piso=x+1,1)), z))



♣ Temp = $\{z' \mid 2 \le 3+z' \le 4,\ z' \ge 0\} = \{0,1\}$, $\Rightarrow$ Temp' = $\{z > 1\}$

♥ Temp = $\{z' \mid 2 \le 3+z' \le 4,\ z' > 1\} = \{\}$, $\Rightarrow$ Temp' = $\{z > 1\}$

A continuación vamos a evaluar el árbol



Hemos llegado al nodo raíz sin marcar, por lo que la interpretación dada satisface la fórmula en el instante 3. Con lo que podemos decir que es la historia dada cumple con la especificación.

Demostración para el algoritmo de History-Checking

La idea se basa en probar el siguiente enunciado:

Sea A una fórmula bien formada y cerrada cualquiera de TRIO' de longitud finita. Entonces una interpretación adecuada dada:

1) satisface A en el instante i (S_i (A) = true) sii el algoritmo comenzando con (A,i,{},T) es exitoso.

2) no satisface A en el instante i (S_i (A) = false) sii el algoritmo comenzando con (A,i,{},F) es exitoso.

La demostración se desarrolla haciendo inducción sobre la estructura de una fórmula TRIO'. Igual que en la demostración anterior el caso base se prueba por inducción sobre la cantidad de variables TD.

Esta demostración se detalla en forma completa en el apéndice C.

Complejidad

Como se dijo anteriormente, este algoritmo construye un árbol de manera similar a un nodo árbol del algoritmo anterior, por lo que se podría tomar parte del cálculo anterior de la complejidad. Por lo tanto se puede concluir que el orden es : **O(mti^n)**, donde n es la longitud de la fórmula de entrada y mti es el mayor entre todos los cardinales de los dominios de las variables TI (no infinitas) y el cardinal de T_f mas 1.

COMENTARIOS SOBRE LOS PROTOTIPOS DESARROLLADOS

Se han implementado los prototipos de los algoritmos propuestos, demostrando efectivamente su factibilidad.

Los mismos se han desarrollado en Pascal for Windows, para aprovechar la modularización ofrecida por las units y el uso de la memoria superior.

Existen unidades que comparten los 2 algoritmos. La unidad *terminos* es la encargada de tratar las fórmulas y sus términos en forma sintáctica. La unit *conjunto* implementa las operaciones de conjuntos definidos por comprensión.

Cada uno de los prototipos tienen una unidad destinada a implementar la estructura correspondiente, es decir un frame para la generación de modelos, en la unit *frame*, y una interpretación para el de history-checking, en la unit *structur*. Estas unidades son fácilmente modificables para adecuarlas a distintas necesidades. Dos ejemplos componen el prototipo de history-checking, cada uno con distinta temporal structure, siendo uno de los mismos con la temporal structure, y la fórmula ofrecida como primer prueba, del ejemplo del ascensor.

El prototipo de generación de modelos posee una unit denominada *arboles* que representa las estructuras del árbol principal y de los subárboles. Para construir y evaluar los árboles se ejecutan dos unidades llamadas *construc* y *evaluar* respectivamente.

Por su parte el otro prototipo posee, análogamente, la unit *nodos* que representa la estructura del árbol y sus nodos necesarios para tal algoritmo. La unit *algoritm* realiza la construcción y evaluación de dicho árbol.

Ambos programas reciben como entrada la fórmula y el instante de tiempo que se quiere evaluar.

Como resultado, el history-checking devuelve el árbol construido y si se pudo, o no, satisfacer la fórmula. El otro prototipo devuelve todos los árboles generados y los valores obtenidos para los predicados y variables TD, indicando también si el resultado ha sido exitoso o no. Los nodos marcados en ambos casos son mostrados.

Para ejecutar los programas se deberá escribir la fórmula como lo indica la pantalla, es decir sin blancos en el medio, con los siguientes símbolos, utilizados como palabras reservadas:

- # : cuantificador universal,

- & : And,
- - : Not,
- % : Dist,
- () Separador

Además la variable, al lado del cuantificador no lleva ().

Ejemplo: La fórmula $\forall x$ (Dist(A(x),m+2) $\wedge \neg$ B(x)) se deberá escribir como

#x(%(A(x),m+2)&-B(x))

Como se verá en el ejemplo, el & es un operador binario y la fórmula se escribe de manera natural. Se han elegido esos símbolos para que el usuario los ingrese desde cualquier teclado.

La fórmula debe estar bien escrita, ya que el programa no realiza ningún chequeo sintáctico, debido a que el mismo es un prototipo para demostrar una real implementación del algoritmo propuesto y no un programa comercial.

CONCLUSIONES, TRABAJOS RELACIONADOS Y FUTUROS

Hemos comparado distintas aproximaciones para tratar el problema del infinito y hemos visto que todas han sido inadecuadas tanto semántica como algoritmicamente.

Se ha definido TRIO' basado en TRIO, como una lógica temporal lineal de primer orden, cuya nueva sintaxis y semántica han permitido corregir todos estos inconvenientes, como se ha demostrado en los capítulos precedentes.

El concepto de variable infinita introducido en el presente trabajo, cumple con el propósito de representar todo el espacio de tiempo en forma infinita. Su uso no ha sido exhaustivo aquí, ya que lo que se pretende es aportar una base para futuras investigaciones, de las que seguramente tendrán el concepto de variable infinita como un importante punto de partida. Como por ejemplo su incorporación a otras lógicas, a otros dominios o en otras partes de las fórmulas.

El lenguaje presentado puede ser utilizado como un kernel de un completo entorno de desarrollo de sistemas. Al mismo modo que TRIO, TRIO' puede ser utilizado para ser la base de lenguajes más estructurados, incluso con mecanismos de abstracción y modularización, como por ejemplo TRIO+ o ASTRAL. TRIO+ [MSP91], es una extensión orientada a objetos que permite organizar los objetos en clases, heredar relaciones entre las clases proveer herencia y generalidad para soportar reusabilidad y desarrollos top-down. Además provee una interface gráfica para representar clases y objetos. Este desarrollo permite generar casos de simulación directamente sobre la representación TRIO+; Astral es otro ejemplo donde el sistema es visto como una colección de máquinas abstractas que se pueden comunicar unas con otras a través de la exportación de variables, Astral está definido formalmente como una traducción en TRIO, lo que permite el uso de las herramientas definidas para este. El análisis de la composición de partes ya verificadas es una área de gran auge en la investigación actual, para tratar por ejemplo con modularización y abstracción. Se podrían plantear nuevos tratamientos estudiando la intersección de los modelos obtenidos.

En [CCMSP93] se define el dominio temporal como la unión de conjuntos disjuntos de distinta granularidad. Esto permite especificar en un mismo sistema distintos grados de tiempo (por ejemplo minutos, segundos, días, meses, etc.). Luego se define un conjunto de reglas semánticas y sintácticas para tratar con estos dominios, pero en ningún caso se permite el uso de dominios infinitos. Se

podría pensar en una extensión del artículo referido, agregando variables infinitas para algunos subconjuntos del dominio temporal.

En este trabajo, además de definir el formalismo, se han desarrollado y demostrado algoritmos para tratar las variables infinitas en forma decidible. El algoritmo de generación de modelos permite verificar la consistencia de especificaciones, probar propiedades y realizar casos de simulación generando eventuales historias de una especificación dada. Por su parte, el algoritmo de history-checking chequea una historia del sistema con respecto a una especificación que se supone la ha generado.

Como ejemplos de los algoritmos propuestos se desarrollaron implementaciones para cada uno. Estas implementaciones no pretenden ser un producto final, sino solo son un prototipo para observar la factibilidad de cada algoritmo, debido a que por su complejidad todavía no han llegado a ser eficientes, ya que son la primer versión de un nuevo formalismo. Por lo que reducir su complejidad plantea un interesante trabajo.

Como se ha notado, varias líneas de investigación pueden ser derivadas a partir del presente trabajo debido a las importantes novedades respecto al tratamiento tradicional de la lógica temporal.

Apéndice A

En este apéndice se desarrollarán todas las demostraciones del capítulo 3.

Lema 1.1 : Los esquemas de axiomas, Ax_1,Ax_2 y Ax_3, de tal lógica son válidos.

Demostración:

Debido a que el valor semántico del $\neg$ y del $\wedge$ son los mismos que para la lógica proposicional, entonces el valor del operador $\rightarrow$ es también el usual, por lo que las tablas de verdad de los esquemas dan como resultado siempre el valor True, para cualquier instante de tiempo de cualquier interpretación en que se evalúe, por lo tanto vale.

Lema 1.2 : Los axiomas de tal lógica son válidos.

Demostración :

Sea A un esquema de axioma Axi, con letras de predicados $a_1,.....,a_n$. Si reemplazamos $a_1,.....,a_n$ en A, por las fórmulas $A_1,.....,A_n$, la nueva fórmula obtenida es válida ya que para cualquier asignación de valores que tomen las fórmulas reemplazantes, y por lo tanto para cualquier valor a las letras de predicados de A, la evaluación es True para todo instante de tiempo, por lema 1.1.

Por lo tanto todos los axiomas son válidos.

Lema 1.3 : La Regla MP conserva la validez lógica. Es decir si A y B son 2 fórmulas cualquiera, A y A $\rightarrow$ B son válidas, entonces B es válida.

Demostración:

Como dijimos anteriormente, la definición del $\wedge$ es en la forma usual y el $\rightarrow$ fue definido en función del $\wedge$ de forma también estándar, por lo tanto se mantiene la tabla de verdad habitual.

Supongamos que B no es True en el instante $j \in T\infty$ de una interpretación I cualquiera, entonces $S_j(B)$ = False. Por hipótesis $S_j(A)$ = True por ser válida, entonces $S_j(A \rightarrow B)$ = False. ABSURDO, que vino de suponer que existe j tal que $S_j(B)$ = False, por lo tanto $S_j(B)$ = True para todo $j \in T\infty$ de cualquier I, entonces B es válida.

Proposición : Con las definiciones anteriores para el $\neg$ y el $\wedge$, todos los teoremas de la lógica proposicional, y por lo tanto todas las tautologías, son válidas en TRIO'.

Demostración:

Si A es un teorema cualquiera de la lógica proposicional, entonces, existe una prueba : $d_1,.....,d_n=A$ tal que $1<= i <=n$:

d_i es un axioma ó

d_i se deduce de d_j y d_k por MP, $j<i$, $k<i$.

Veamos que cada d_i es válido, haciendo inducción por la longitud de la prueba.

Caso base: $n=1$

$d_1=A$ es un axioma, entonces por lema 1.2 A es Válido.

Caso inductivo: Vale para todo d_i , $i<n$

** Si d_n es un axioma entonces por lema 1.2 d_n es válido.*

** Si d_n se dedujo de 2 d_z y d_x ($z<n$ y $x<n$), entonces por hipótesis inductiva d_z y d_x son Válidos, entonces por lema 1.3 d_n es válido.*

Por lo tanto todo teorema de la lógica proposicional es válido, por lo que, usando el teorema de corrección [Men79], toda tautología es válida.

Corolario : Todas las contradicciones son False para cualquier instante de tiempo de cualquier interpretación.

Demostración:

Supongamos que A es una contradicción, entonces por la definición del $\neg$ dada, $\neg A$ es una tautología, entonces por proposición anterior $\neg A$ es válida, entonces $\forall j$ de toda interpretación $S_j(\neg A) = True$, por lo que por la tabla del $\neg$, $S_j(A) = False$ $\forall j$ de toda interpretación, por lo que quedo demostrado.

Proposición : Sea A un teorema de la lógica proposicional, entonces Always(A) es válida.

Demostración:

Se definió Always (A) $\equiv$ $\forall t$ (Dist (A, t)), $\forall t \in T\infty$, por lo tanto, sea S_j una evaluación cualquiera de una interpretación I cualquiera, por la Proposición anterior $S_j(A) = True$, $\forall j \in T\infty$, entonces dado que S_i (Dist (A, t)) = $S_k(A)$, $k= i+S_i(t)$ y $k \in T\infty$ $\forall t$, entonces $S_k(A) = True$, $\forall k$, o lo que es lo mismo S_i (Dist (A, t)) = True $\forall t \in T\infty$, $T\infty \in I$ para cualquier I, por lo que Always(A) es válida.

Proposición : Vale la Temporal Transparency, o sea Dist ($\neg$ A , t) $\equiv \neg$ (Dist (A , t)).

Demostración:

Sea S_i una evaluación cualquiera:

S_i ***(Dist ($\neg$ A , t))*** $\equiv S_{i+v}$ *($\neg$ A),* $v = S_i(t) \equiv$

$\equiv \neg S_{i+v}$ *(A),* $v = S_i(t) \equiv \neg$ ***S_i (Dist (A , t))***

Corolario : Always($\neg$A) $\equiv \neg$ Always (A)

Proposición : Vale Dist (A $\wedge$ B , t) $\equiv$ (Dist (A , t)) $\wedge$ (Dist (B , t)).

Demostración:

Sea S_i una evaluación cualquiera:

S_i (Dist (A $\wedge$ B , t)) $\equiv S_{i+v}$ *(A $\wedge$ B),* $v = S_i(t) \equiv$

S_{i+v} *(A)* $\wedge$ S_{i+v} *(B),* $v = S_i(t)$ $\equiv$***S_i (Dist (A , t)) $\wedge$ S_i (Dist (B , t))***

Corolario : Vale : Always (A $\wedge$ B) $\equiv$ Always (A) $\wedge$ Always (B)

Corolario : Si valen Always (A) y Always (A $\rightarrow$ B), entonces vale Always (B)

Demostración:

Demostraremos por el absurdo:

Supongamos que existe j $\in T\infty$, tal que S_j (B) = False. Por hipótesis S_i ($\forall$z Dist(A,z)) = True, z $\in T\infty$, en particular para j S_j (A) = True, análogamente S_i ($\forall$z Dist(A$\rightarrow$B,z)) = True, z $\in T\infty$, y S_j (A$\rightarrow$B) = True,

con lo que S_j (B) = True, ABSURDO, que vino de suponer que existe j $\in T\infty$, tal que S_j (B) = False, por lo que S_i ($\forall$z Dist(B,z)) = True,z $\in T\infty$, con lo que concluimos que vale Always (B).

Proposición : Vale : Always (A) $\rightarrow$ Sometimes (A)

Demostración:

Dado que S_i($\forall$z Dist(A,z)))=True, z $\in T\infty$, entonces en particular existe a $\in$ D(z), tal que S_i(Dist(A,a)))=True, o sea S_i($\exists$z Dist(A,z)))=True, entonces Sometimes (A) es True. Por lo que (Always (A) $\rightarrow$ Sometimes (A)) vale.

Proposición : Si i,j $\notin T_f$ y t es un término cualquiera, entonces $S_i(t)=S_j(t)$

Demostración :

Caso Base :

Si x variable TI, entonces $S_i(x) = \xi(x) = S_j(x)$.

Si x variable TD, entonces como i,j $\notin T_f$ $S_i(x) = \eta_e(x) = S_j(x)$.

Si f es una función 0-aria, entonces $S_i(f) = \Phi(f) = S_j(f)$.

Por lo que vale para el caso base.

Caso Inductivo :

Si f es una función n-aria, entonces $S_i(f(t_1,....,t_n)) = \Phi(f)(S_i(t_1),.....,S_i(t_n))$ = (por hipótesis inductiva) $\Phi(f)(S_j(t_1),.....,S_j(t_n)) = S_j(f(t_1,....,t_n))$

Por lo que vale para todos los términos.

Proposición : Si i,j $\notin T_f$ y A es una fórmula TRIO' que no contiene el operador Dist entonces $S_i(A)=S_j(A)$

Demostración:

Ahora se hará inducción sobre la estructura de una fórmula TRIO'.

Caso Base :

Sea p una fórmula atómica:

Si p es un predicado TI, entonces

$S_i(p(t_1,.....,t_n))$ = true sii $(S_i(t_1),.....,S_i(t_n)) \in \Pi(p)$, sii por la proposición anterior, $(S_j(t_1),.....,S_j(t_n)) \in \Pi(p)$ sii $S_j(p(t_1,.....,t_n))$ = true.

Si p es un predicado TD, dado que i,j $\notin T_f$, entonces

$S_i(p(t_1,.....,t_n))$ = true sii $(S_i(t_1),.....,S_i(t_n)) \in \Pi_e(p)$, sii por la proposición anterior, $(S_j(t_1),.....,S_j(t_n)) \in \Pi_e(p)$ sii $S_j(p(t_1,.....,t_n))$ = true.

Caso Inductivo :

Si A y B son fórmulas cualquiera :

- *$S_i(\neg A) = \neg(S_i(A)) =$ (por hipótesis inductiva) $\neg(S_j(A)) = S_j(\neg A)$.*
- *$S_i(A \wedge B) = S_i(A) \wedge S_i(B) =$ (por hipótesis inductiva) $S_j(A) \wedge S_j(B) = S_j(A \wedge B)$.*

- *S_i ($\forall(x)$ (A(x))) = AND_{ak} $S_i(A(^x/_{ak}))$, para todo $a_k \in D(x)$ = (por hipótesis inductiva) AND_{ak} $S_j(A(^x/_{ak}))$, para todo $a_k \in D(x)$ = S_j ($\forall(x)$ (A(x))).*

Con lo que queda demostrada la proposición.

Corolario : Las siguientes propiedad es válida: $\neg \neg$ A -> A.

Corolario : Si A es una fórmula sin predicados TD y sus términos no contienen variables TD entonces

☑ Dist (A, t) ≡ A

☑ $S_i(A)=S_j(A)$, $\forall i,j \in T\infty$.

Las demostraciones de estos últimos corolarios son directas.

Corolario : No vale Dist (Dist (A , t_1) , t_2) ≡ Dist (A , $t_1 + t_2$).

Demostración:

Sea la siguiente interpretación:

- *Variables TD: m: η_1 (m) = 1 , η_2 (m) = 2 , η_e (m) = 2*
- *La función de suma es asumida en la forma usual.*
- *Dominios: D(m)=[1..11]*
- *T_f=[2..40] , T∞ = Z^+.*
- *Predicados TD A : Π_3(A)={( )} , Π_4(A)={}*

Sean t_1=m y t_2=1 entonces:

*S_1 (Dist (Dist (A , t_1) , t_2)) = S_2 (Dist (A , t_1)), [2=1+$S_1(t_2)$=1+1] ≡ S_4 (A), [4=2+$S_2(t_1)$=2+2] =**False***

*S_1 (Dist (A , $t_1 + t_2$)) = S_3 (A),[3=1+$S_1(t_1+t_2)$ =1+$S_1(t_1)$+$S_1(t_2)$=1+1+1] =**True.***

Apéndice B

Este apéndice desarrolla la demostración formal del algoritmo de Generación de Modelos.

Los 3 siguientes lemas que servirán para la demostración principal del algoritmo.

• Lema 1 : Si en un nodo (F,i,asg,V) hay más de una variable infinita que esté libre en la fórmula F o que esté en el instante de tiempo i entonces es la misma variable.

Demostración :

Supongamos que :

i) La fórmula F tiene más de una variable infinita libre, entonces por construcción F es una subfórmula de la fórmula de nodo raíz, que por ser cerrada deja a la subfórmula F dentro del alcance de los cuantificadores de las distintas variables, por lo tanto al menos una de las variables infinitas, va a estar anidada. Absurdo, pues la fórmula del nodo raíz es también fórmula bien formada.

ii) En el instante de tiempo hay más de una variable infinita distinta, entonces en algún nodo ascendente hay una fórmula Dist(A, t), donde F está en A y t en i, ya que es la única forma que asigna variables al instante de tiempo, donde :

- t tiene al menos dos variables infinitas libres, por lo que estamos en el caso i).

- t tiene una sola variable infinita libre, entonces Dist(A,t) es una subfórmula de otra Dist(B,j) donde j tiene otra variable infinita libre, por lo que Dist (B,j) tiene dos variables infinitas libres distintas, por lo que estamos en el caso i).

iii) La fórmula F tiene una variable infinita libre e i tiene otra en su instante de tiempo, entonces ya que la variable del instante de tiempo viene de un nodo del tipo Dist(A,t), donde la variable infinita está en t y donde la fórmula F está contenida en A entonces Dist(A,t) tiene dos variables infinitas libres distintas, por lo que estamos en el caso i)

Entonces como i), ii) y iii) cubren todos los casos posibles, el lema es válido.

• Lema 2 : Si en dos nodos hijos de un nodo de la forma (B∧C,i,asg,V) hay variables infinitas libres en las fórmulas o en el instante de tiempo de cada hijo entonces son las mismas variables.

Demostración:

Dado que los dos algoritmos no modifican el instante de tiempo para los nodos de la fórmula con la forma mencionada, entonces todos los hijos tienen el mismo instante de tiempo que el nodo padre, por lo que por lema 1 tiene a lo sumo una única variable infinita, entonces la única alternativa que queda es que las variables distintas estén en las fórmulas de cada nodo hijo, o sea en B y en C, entonces la fórmula A tendría dos variables infinitas libres, Absurdo también por lema 1:

• Lema 3 : Si en los nodos hijos de un nodo de la forma (∀xA,i,asg,V) hay variables infinitas libres en las fórmulas o en el instante de tiempo de cada hijo entonces son las mismas variables.

Demostración:

Dado que los nodos hijos son descendientes de un nodo cuya fórmula es ∀xA, estos solo difieren en la fórmula A con la variable x instanciada, por lo que no podrían tener variables infinitas distintas, ni en la fórmula ni en los instantes de tiempo, sino los tendría el nodo padre, Absurdo por lema 1.

• Corolario1: Si x es una variable infinita, entonces el nodo hijo de un nodo de la forma (∀xA,i,asg,V) tiene como única variable infinita libre a x y no tiene variable infinita en i.

Demostración:

El nodo hijo es de la forma (A, i, asg, V) donde x está libre en A, entonces por Lema 1 es la única y no hay variable infinita en i.

A continuación se prueba que el algoritmo busca al menos un modelo de la fórmula dada como entrada.

Sea A una fórmula bien formada y cerrada cualquiera de TRIO'. Entonces existe una interpretación (temporal structure), construida con los valores dados por el algoritmo (asgTD, P_i, P'_e, etc.) y con el frame dado tal que :

1) satisface A en el instante i (S_i (A) = True) sii el algoritmo comenzando con (A,i,T,{}) es exitoso,

2) no satisface A en el instante i (S_i (A) = False) sii el algoritmo comenzando con (A,i,F,{}) es exitoso,

3) Además, si el nodo (F, i, asg, V) no está marcado, F tiene una variable infinita x libre o que está en i, entonces asg indica los valores que toma la variable infinita para que se cumpla el valor de verdad que indica el nodo.

Antes de comenzar con la demostración se deberá recordar que los únicos subárboles que se evalúan son las hojas.

Demostración:

Sea (A,i,V,asg) la raíz de un subárbol, A es una fórmula, se demuestra que se cumple 1) , 2) y 3). Para esto se hace inducción sobre la estructura de una fórmula TRIO' y en el caso base, en particular se hace inducción sobre la cantidad de variables TD.

Caso Base sobre la estructura de la fórmula (A es una fórmula atómica):

A es una fórmula atómica, por ser cerrada es ground o solo tiene variables TD.

Caso base: A es una fórmula atómica $p(v_1,.....,v_n)$, ground,

- *La construcción del subárbol no hace nada.*
- *La parte de evaluación hace:*

• *p es un predicado TI entonces:*

Si V=T

La tupla de valores $(v_1,.....,v_n) \notin P(p)$ sii el nodo es marcado. Por lo tanto

$\exists S_i$ tal que $S_i(A)$ = True sii la tupla de valores $(v_1,.....,v_n) \in P(p)$ sii el nodo no es marcado sii el algoritmo es exitoso.

Si V=F

La tupla de valores $(v_1,.....,v_n) \in P(p)$ sii el nodo es marcado. Por lo tanto

$\exists S_i$ tal que S_i (A) = False sii la tupla de valores $(v_1,.....,v_n) \notin P(p)$ sii el nodo no es marcado sii el algoritmo es exitoso.

• *p es un predicado TD entonces:*

Si V=T

- *Si $i \in T_f$, entonces el algoritmo, ya que $(v_1,.....,v_n) \notin P'_i(p)$, por ser este el único nodo del árbol, hace:*

  $P_i(p) = P_i(p) \dot{E} \{(v_1,.....,v_n)\}$ y termina con éxito. Luego la tupla de valores $(v_1,.....,v_n) \in P_i(p)$ sii $\exists S_i$ tal que $S_i(A)$= True.

- *Si $i \notin T_f$, entonces el algoritmo, ya que $(v_1,.....,v_n) \notin P'_e(p)$, por ser este el único nodo del árbol, hace:*

  $P_e(p) = P_e(p) \dot{E} \{(v_1,.....,v_n)\}$ y termina con éxito. Luego la tupla de valores $(v_1,.....,v_n) \in P_e(p)$ sii $\exists S_i$ tal que $S_i(A)$ = True.

Si V=F

- *Si $i \in T_f$, entonces el algoritmo, ya que $(v_1,.....,v_n) \notin P_i(p)$, por ser este el único nodo del árbol, hace:*

$P'_i(p) = P'_i(p) \dot{E} \{(v_1,.....,v_n)\}$ y termina con éxito. Luego la tupla de valores $(v_1,.....,v_n) \notin P_i(p)$ sii $\exists S_i$ tal que $S_i(A) = False$.

• Si $i \notin T_f$, entonces el algoritmo, ya que $(v_1,.....,v_n) \notin P_e(p)$, por ser este el único nodo del árbol, hace:

$P'_e(p) = P'_e(p) \dot{E} \{(v_1,.....,v_n)\}$ y termina con éxito. Luego la tupla de valores $(v_1,.....,v_n) \notin P_e(p)$ sii $\exists S_i$ tal que $S_i(A) = False$.

- El algoritmo para una fórmula atómica ground no asigna nada, ni a asg, ni a asgtd, por lo tanto dado que no tiene ninguna variable infinita ni TD, vale para cualquier asg y asgTD, en particular para los obtenidos por el algoritmo.

✧ Caso inductivo: A es una fórmula atómica con n variables TD.

Supongamos que z es la n-esima variable TD y que vale para las n-1 variables TD restantes.

- La construcción del árbol hace:

Si ya hubiese habido alguna asignación para z, entonces z se hubiese reemplazado en F, con lo que z no sería entonces la n-esima variable TD, por lo tanto $(z , i, b) \notin asgTD$, entonces se generan $|D(z)|$ subárboles hijos, cada uno con $(F(^{z}/_{a}),i,\{},V)$, $a \in D(z)$ y $asgTD = asgTD \dot{E} \{(z,i,a)\}$.

- La parte de evaluación hace:

✧ Si V=T

• Si el algoritmo es exitoso, entonces, en algún subárbol, su hijo no está marcado por lo tanto, por hipótesis inductiva $\exists$ Si tal que Si (A(z/a)) = True, con lo que Si(A)= True para el valor a de la variable TD z en el instante i ($(z,i,a) \in asgTD$), con lo que Si(A)= True para $h_i(z)=a$.

• El algoritmo no es exitoso, entonces, en todo subárbol, su hijo está marcado, entonces por hipótesis inductiva no $\exists a \in D(z)$ tal que Si (A(z/a)) es True con lo que Si (A) no es True para ningún valor de z, por lo que no $\exists$ Si tal que Si (A) es True.

✧ Si V=F

• Si el algoritmo es exitoso, entonces, en algún subárbol, su hijo no está marcado por lo tanto, por hipótesis inductiva $\exists S_i$ tal que S_i ($A(^{z}/_{a})$) = False, con lo que $S_i(A)$= false para el valor a de la variable TD z en el instante i ($(z,i,a) \in asgTD$), con lo que $S_i(A)$= false para $h_i(z)=a$.

• El algoritmo no es exitoso, entonces, en todo subárbol, su hijo está marcado, entonces por hipótesis inductiva no $\exists a \in D(z)$ tal que S_i ($A(^{z}/_{a})$) es False con lo que S_i (A) no es False para ningún valor de z, por lo que no $\exists S_i$ tal que S_i (A) es False.

Igual que en el caso anterior el algoritmo no asigna nada a asg, pero como no tiene variable infinita, entonces vale 3) para cualquier asg.

Por lo que podemos concluir que vale 1) , 2) y 3) para fórmulas atómicas cerradas.

Caso inductivo sobre la estructura de la fórmula:

➢ *A es de la forma ¬B*

⊕ *V=T*

- La construcción del subárbol crea un nodo hijo con (B,i,asg, F).

- La parte de evaluación hace:

• Si no es exitoso el algoritmo entonces, en todo subárbol, el nodo hijo está marcado. Por lo tanto, para cualquier subárbol, si el nodo con (B,i,asg,F) está marcado, por hipótesis inductiva no $\exists S_i$ *tal que se cumple que* $S_i(B)$ *= False, por lo tanto* $S_i(B)$ *= True con lo que* $S_i(\neg B)$ *= False, con lo que no* $\exists S_i$ *tal que* $S_i(A)$ *= False.*

• En caso contrario, en algún subárbol, el algoritmo solo copia asg. Entonces como no estuvo marcado el nodo hijo para (B,i,asg,F), por hipótesis inductiva $\exists S_i$ *tal que se cumple que* $S_i(B)$ *= False, por lo tanto* $S_i(\neg B)$ *= True con lo que* $\exists S_i$ *tal que* $S_i(A)$ *= True. (i)*

⊕ *V=F*

La construcción del subárbol crea un nodo hijo con (B,i,asg, T).

La parte de evaluación hace:

• Si el algoritmo no es exitoso, para todo subárbol, el nodo hijo está marcado. Por lo tanto para cualquier subárbol, si el nodo con (B,i,asg,T) está marcado, por hipótesis inductiva no $\exists S_i$ *tal que se cumple que* $S_i(B)$ *= True, por lo tanto* $S_i(B)$ *= False con lo que* $S_i(\neg B)$ *= True, con lo que* $\exists S_i$ *tal que* $S_i(A)$ *= True.*

• En caso contrario, en algún subárbol, el algoritmo solo copia asg. Entonces como no está marcado el nodo para (B,i,asg,T), por hipótesis inductiva se cumple que $\exists S_i$ *tal que* $S_i(B)$ *= True, por lo tanto* $S_i(\neg B)$ *= False con lo que* $\exists S_i$ *tal que* $S_i(A)$ *= False.(ii)*

- Ya que (i) y (ii) no modifican los valores de las variables infinitas, y por hipótesis inductiva, asg indica los valores que toma la variable infinita libre para el nodo hijo, entonces vale 3) para el mismo asg, por lo tanto vale 3) para este caso.

Por consiguiente vale 1) , 2) y 3) para A=¬B.

➢ *Si A=B ∧ C:*

- La primer parte del algoritmo crea dos nodos hijos con (B,i,asg,V) y con (C,i,asg,V) cada uno.

- La parte de evaluación hace:

⊕ *Si V=T*

• Si el algoritmo es exitoso, entonces en algún subárbol, ninguno de sus hijos está marcado por lo tanto, por hipótesis inductiva $\exists S_i$ *tal que* S_i *(B) = True y* S_i *(C) = True, con lo que* $S_i(A)$*= True.(iii)*

• *El algoritmo no es exitoso, entonces, en todo subárbol, al menos uno de sus hijos está marcado, entonces por hipótesis inductiva no* $\exists S_i$ *tal que* $S_i(B) = True$ *o* $S_i(C) = True$ *con lo que no* $\exists S_i$ *tal que* $S_i(A) = True$.

Si V=F

• *Si el algoritmo es exitoso, entonces, en un subárbol, al menos uno de sus hijos no está marcado, supongamos el nodo que contiene la subfórmula B, por lo tanto por hipótesis inductiva* $\exists S_i$ *tal que* $S_i(B)$ *es False, por lo tanto,* $\exists S_i$ *tal que* $S_i(A)$ = *False. Análogamente si el nodo no marcado es el que contiene la subfórmula C, (iv)*

• *El algoritmo no es exitoso, entonces en todo subárbol, todos sus hijos están marcados, entonces por hipótesis inductiva no* $\exists S_i$ *tal que* $S_i(B) = False$ *ni* $S_i(C) = False$ *con lo que no* $\exists S_i$ *tal que* $S_i(A)$ *es False.*

-En (iii) y (iv) por hipótesis inductiva asg de los nodos hijos indican los valores que toman las variables infinitas libres o las que están en i, de cada uno, llamemos asg_B *al asg obtenido en el nodo (B,i,asg,V) y análogamente* asg_C *para (C,i,asg,V).entonces*

- *Supongamos que ni la fórmula A contiene variables infinitas libres, ni i contiene variable infinita, por lo por construcción ni B, ni C contienen variables infinitas libres, ni i de los nodos hijos, entonces no se necesitan valores de asg, o sea para asg no se asigna nada como el algoritmo dice.*
- *En cada nodo hay una variable infinita que cumpla con esta condición, entonces por lema 2 es la misma, por lo que A vale : en (iii) para los mismos valores de la variable que están en los dos nodos, o sea para* $asg_B \cap asg_C$ *y en (iv) para cualquier valor de la variable que está en los dos nodos, o sea para* asg_B *È* asg_C *, o sea como el algoritmo dice.*
- *Hay una variable infinita que cumpla la condición en un solo nodo, entonces A vale para los mismos valores de la variable que están en ese nodo, ya que no interesa el asg del nodo hermano, pues no tiene variable infinita libre (por construcción tampoco i), o sea como el algoritmo dice.*

Por lo tanto vale 1), 2) y 3) para este caso.

Si $A = \forall x\, B$ *:*

• *x no es la variable infinita:*

- La primer parte del algoritmo crea | D(x) | nodos hijos, cada uno con $(B^x/_a, i, asg, V)$, $\forall a \in D(x)$.

- La parte de evaluación hace:

Si V=T

• *Si el algoritmo es exitoso, entonces, en al menos un subárbol, ninguno de sus nodos hijos está marcado, por lo tanto, por hipótesis inductiva* $\exists S_i$ *tal que* $S_i(B^x/_a)$ *es True para todo* $a \in D(x)$, *con lo que* $S_i(\forall x\, B) = True$, *por consiguiente* $S_i(A) = True$. *(v)*

• *El algoritmo no es exitoso, por lo tanto en todo subárbol, alguno de sus hijos está marcado, entonces por hipótesis inductiva no $\exists S_i$ tal que S_i ($B^x/_a$) es True para algún $a \in D(x)$, con lo que no $\exists S_i$ tal que S_i ($\forall x B$) es True, entonces no $\exists S_i$ tal que S_i (A) es True.*

Si V=F

• *El algoritmo es exitoso, entonces al menos uno de sus nodos hijos no está marcado, digamos el que asigna a x el valor a1, entonces por hipótesis inductiva $\exists S_i$ tal que S_i ($B^x/_{a1}$) es False para $a1 \in D(x)$, por lo tanto, $S_i(\forall x B)$ = False, por consiguiente $\exists S_i$ tal que $S_i(A)$ = False. (vi)*

• *El algoritmo no es exitoso, entonces, todos sus nodos hijos están marcados, entonces para todo $a_i \in D(x)$, el nodo que contiene la tupla ($B^x/_{ai}$,i,asg,F) está marcado, entonces por hipótesis inductiva no $\exists S_i$ tal que S_i ($B^x/_{a1}$) es False para ningún a_i , con lo que no $\exists S_i$ tal que S_i ($\forall x B$) es False, por consiguiente no $\exists S_i$ tal que $S_i(A)$ es False.*

- Supongamos que ni la fórmula A contiene variables infinitas libres, ni i contiene variables infinitas, entonces no se necesitan valores de asg, o sea para asg no se asigna nada como el algoritmo dice.

Supongamos, ahora el caso contrario, en (v) y (vi) por hipótesis inductiva, el conjunto asg de cada nodo hijo indica los valores que toman las variables infinitas libres o que están en i, del mismo entonces por lema 3, los nodos hijos tienen la misma y única variable infinita libre o en i, por lo que, entonces para el nodo padre los valores de la variable infinita son : en (v) los mismos valores de la variable infinita que están en todos los nodos hijos, o sea para la intersección de los mismos y en (vi) para cualquier valor de la variable infinita que está en los nodos hijos válidos, o sea para la unión de los mismos que no están marcados, o sea como el algoritmo dice.

• *x es la variable infinita:*

- La primer parte del algoritmo crea un nodo hijo con (B ,i ,asg ,V).[XVI]

- La parte de evaluación hace:

Si V=T

• *Si el algoritmo es exitoso, entonces en algún subárbol, su nodo hijo no está marcado y asg del mismo es igual a D(x), por lo tanto, por hipótesis inductiva $\exists S_i$ tal que S_i (B) es True para todo valor de la variable infinita que está en asg, que por Corolario 1 es x, o sea para todo valor de x, con lo que $S_i(\forall x B)$= True, por consiguiente $\exists S_i$ tal que $S_i(A)$= True.*

• *El algoritmo no es exitoso, por lo tanto en todo subárbol, o su hijo está marcado o asg del mismo es distinto de D(x) entonces por hipótesis inductiva, $\forall S_i$, S_i (B)= False o S_i ($B^x/_a$) no es True para algún $a \in D(x)$ y $a \notin$ asg, con lo que no $\exists S_i$ tal que S_i ($\forall x B$) es True, por consiguiente no $\exists S_i$ tal que S_i (A) es True.*

Si V=F

• *Si el algoritmo es exitoso, entonces en algún subárbol, su nodo hijo no está marcado y asg del mismo es distinto del conjunto vacío, por lo tanto, por hipótesis inductiva $\exists S_i$ tal que S_i (B) es False para algún*

[XVI] Notar que en B, x está libre.

valor de la variable infinita, que por Corolario 1 es x, que ∈ a D(x), con lo que $S_i(\forall x\ B)$= False, por consiguiente $\exists S_i$ tal que $S_i(A)$= False para el mismo asg.

• El algoritmo no es exitoso, por lo tanto en todo subárbol, o su hijo está marcado o asg del mismo es igual a vacío ($\forall a \in D(x)$, $a \notin asg$), entonces por hipótesis inductiva, no $\exists S_i$ tal que o S_i (B) es False o $S_i(B^x/_a)$= False, con lo que S_i ($\forall x$B) no es False, por consiguiente no $\exists S_i$ tal que S_i (A)es False.

- En los casos que el algoritmo sea exitoso, no se necesita copiar asg ya que la única variable que está libre en el nodo hijo queda ligada en el nodo padre y por lema 1 es la única, ya que sino el nodo hijo tendría más de una variable libre.

Por lo tanto vale 1) , 2) y 3) para este caso.

➢Si A = Dist(B,t):

• t no tiene ninguna variable:

⌑ Si | asg | ≤ 1

- La primer parte del algoritmo obtiene k como $i+S_i(t)$ y crea un nodo hijo con (B, k , asg, V).

- En la parte de evaluación:

⊕ V=T

• El algoritmo es exitoso, entonces en algún subárbol el nodo hijo no está marcado, entonces por hipótesis inductiva $\exists S_i$ tal que $S_k(B)$ es True para $k=i+S_i(t)$, con lo que $\exists S_i$ tal que $S_i(Dist(B,t))$ es True, por lo tanto $S_i(A)$ es True.(vii)

• El algoritmo no es exitoso, por lo tanto en todo subárbol, el nodo hijo está marcado, entonces por hipótesis inductiva no $\exists S_i$ tal que $S_k(B)$ es True para $k=i+S_i(t)$, por lo tanto no $\exists S_i$ tal que $S_i(Dist(B,t))$ es True, con lo que no $\exists S_i$ tal que $S_i(A)$ es True.

⊕ V = F

• El algoritmo es exitoso, entonces en algún subárbol, el nodo hijo no está marcado, por lo tanto, por hipótesis inductiva $\exists S_i$ tal que $S_k(B)$ es False para $k=i+S_i(t)$, con lo que $S_i(Dist(B,t))$ es False, por lo tanto $\exists S_i$ tal que Si(A) es False.(viii)

• El algoritmo no es exitoso, por lo que en todo subárbol el nodo hijo está marcado, entonces por hipótesis inductiva no $\exists S_i$ tal que $S_k(B)$ es False para $k=i+S_i(t)$, por lo tanto $S_i(Dist(B,t))$ no es False, con lo que $\exists S_i$ tal que $S_i(A)$ es False.

Ya que (vii) y (vii) no modifican los valores de las variables infinitas, y por hipótesis inductiva, asg indica los valores que toma la variable infinita libre para el nodo hijo, entonces vale 3) para el mismo asg, por lo tanto vale 3) para este caso.

⌑ Si | asg | >1, entonces, i representa una expresión donde la variable infinita toma más de un valor, que están indicados en asg, con lo que puede haber nuevos valores para que el instante de tiempo esté dentro de T_f, por lo que el algoritmo vuelve a calcular los valores de la variable infinita.

- La primer parte del algoritmo hace, si x es la variable infinita que está en i:

temp = { x'/ $S_i(t)+i^x/_{x'} \in T_f$, x' ∈ asg }

temp'= asg - temp

Si temp ≠ {} : Crea | temp | nodos hijos, con (B, $S_i(t)+i^x/_{x'}$,{x'} ,V) , ∀x' ∈ temp.

Crea un nodo hijo con (B, $S_i(t)+i$,temp' ,V).

Es decir si temp = {}, entonces no se volvió a entrar a T_f. Temp' no puede ser vacío ya que, igual que asg, es la diferencia entre un conjunto infinito y uno finito.

- En la parte de evaluación:

⇳ V=T

• El algoritmo es exitoso, entonces en algún subárbol, al menos un nodo hijo no está marcado, entonces por hipótesis inductiva ∃ S_i tal que $S_k(B)$ es True, para k= $S_i(t)+i^x/_{x'}$, x' ∈ asg, con lo que ∃ S_i tal que $S_i(Dist(B,t))$ es True para todos los asg de los hijos sin marcar, por lo tanto ∃ S_i tal que $S_i(A)$ = True, para la unión de los asg de los hijos sin marcar.

• El algoritmo no es exitoso, por lo tanto en todo subárbol, todos los nodos hijos están marcados, entonces por hipótesis inductiva no ∃ S_i tal que $S_k(B)$ es True para k= $S_i(t)+i^x/_{x'}$, x' ∈ asg, por lo tanto no ∃ S_i tal que $S_i(Dist(B,t))$ es True para los valores del asg de los hijos, con lo que no ∃ S_i tal que $S_i(A)$ es True para el asg indicado, que por hipótesis inductiva es el valor que toma la variable infinita, por lo tanto no ∃ S_i tal que $S_i(A)$ es True.

⇳ V = F

• El algoritmo es exitoso, entonces en algún subárbol, al menos un nodo hijo no está marcado, entonces por hipótesis inductiva ∃ S_i tal que $S_k(B)$ es False, para k= $S_i(t)+i^x/_{x'}$, x' ∈ asg, con lo que ∃ S_i tal que $S_i(Dist(B,t))$ es False para todos los asg de los hijos sin marcar, por lo tanto ∃ S_i tal que $S_i(A)$ = False, para la unión de los asg de los hijos sin marcar.

• El algoritmo no es exitoso, por lo tanto en todo subárbol, todos los nodos hijos están marcados, entonces por hipótesis inductiva no ∃ S_i tal que $S_k(B)$ es False para k= $S_i(t)+i^x/_{x'}$, x' ∈ asg, por lo tanto no ∃ S_i tal que $S_i(Dist(B,t))$ es False para los valores del asg de los hijos, con lo que no ∃ S_i tal que $S_i(A)$ es False para el asg indicado, que por hipótesis inductiva es el valor que toma la variable infinita, por lo tanto no ∃ S_i tal que $S_i(A)$ es True.

• Si t contiene solo variables TI y valores ground, entonces crea un nodo hijo con (A , i+t, asg, V)[XVII], *entonces*

Supongamos que x es la variable infinita que está en t,

- La primer parte del algoritmo hace:

[XVII] Notar que, como dijimos en el algoritmo, por ser fórmulas cerradas y por la construcción del árbol, la única variable que puede estar no instanciada a está altura de algoritmo en el término t, es una variable infinita.

temp = { x' / i+$S_i(t^x/_{x'})$ ∈ T_f, x' ∈ D(x) }

temp'= D(x) - temp

Crea | temp | nodos hijos, con (B, i+$S_i(t^x/_{x'})$,{x'} ,V) , ∀x' ∈ temp.

Crea un nodo hijo con (B, i+t,temp' ,V).

- En la parte de evaluación:

⇳ V=T

• El algoritmo es exitoso, entonces en algún subárbol, al menos un nodo hijo no está marcado, entonces por hipótesis inductiva ∃ S_i tal que $S_k(B)$, k=i+$S_i(t^x/_b)$[XVIII], es True para b que está en asg, que por construcción ∈ D(x), o sea para el valor b de la variable infinita, con lo que ∃ S_i tal que $S_i(Dist(B,t))$ es True para el mismo asg y para los asg de los nodos hijos sin marcar, por lo tanto ∃ S_i tal que $S_i(A)$ es True para la unión de los asg de los nodos hijos sin marcar, o sea el asg indicado.

• El algoritmo no es exitoso, por lo que en todo subárbol, todos los nodos hijos están marcados, entonces por hipótesis inductiva no ∃ S_i tal que $S_k(B)$, k=i+ $S_i(t^x/_b)$ es True para ningún b que está en asg de ningún nodo, o sea para ningún b ∈ D(x), por lo tanto no ∃ S_i tal que $S_i(Dist(B,t))$ es True para cualquier valor de la variable infinita, con lo que no ∃ S_i tal que $S_i(A)$ es True.

⇳ V = F

• El algoritmo es exitoso, entonces en algún subárbol, al menos un nodo hijo no está marcado, entonces por hipótesis inductiva ∃ S_i tal que $S_k(B)$ es False, k=i+ $S_i(t^x/_b)$, para b está en asg, que por construcción ∈ D(x), o sea para el valor b de la variable infinita, con lo que ∃ S_i tal que $S_i(Dist(B,t))$ es False para el mismo asg y para los asg de los nodos hijos sin marcar, por lo tanto ∃ S_i tal que $S_i(A)$ es False para la unión de los asg de los nodos hijos sin marcar, o sea el asg indicado.

• El algoritmo no es exitoso, por lo que en todo subárbol, todos los nodos hijos están marcados, entonces por hipótesis inductiva no ∃ S_i tal que $S_k(B)$, k=i+ $S_i(t^x/_b)$ es False para ningún b que está en asg de ningún nodo, o sea para ningún b ∈ D(x), por lo tanto no ∃ S_i tal que $S_i(Dist(B,t))$ es False para cualquier valor de la variable infinita, con lo que no ∃ S_i tal que $S_i(A)$ es False.

• Si t contiene una variable TD z, entonces

- Si (z , i , b) no pertenece a asgtd, para cualquier b ∈ D(z), entonces el algoritmo crea | D(z) | subárboles con un nodo hijo con (Dist(B, $t^z/_a$), i , asg, V) , ∀a ∈ D(z) y con asgtd = asgtd È { (z, i,a) }.

- En caso contrario, crea un solo nodo hijo con (Dist(B, $t^z/_b$), i , asg , V).

En este caso no se puede aplicar inmediatamente la hipótesis inductiva, ya que el solo hecho de asignarle un valor a una variable no modifica la estructura de la fórmula, pero el algoritmo va asignando valores a las variables TD en forma consistente y en un número finito de pasos, ya que se fija si hay alguna asignación anterior y la cantidad de variables TD es finita, por lo que llegamos a los casos anteriores (t sin variables o t con la variable

[XVIII] Notar que S_i ($t^x/_b$) = $t^x/_b$ ya que t solo contiene valores ground.

infinita) que ya han sido demostrados como válidos, por lo que se podrían tomar como los casos base para una inducción sobre el número de variables TD, por lo que se considera este caso como válido.

Por lo tanto vale 1) , 2) y 3) para A = Dist (B , t)

Por lo que vale para el caso inductivo.

Por lo que queda demostrado.

Apéndice C

Para esta demostración se utilizan los lemas 1, 2 y 3 y el Corolario 1, demostrados para el algoritmo de generación de modelos, ya que por ser la construcción similar para ambos algoritmos, no se afectan las demostraciones desarrolladas.

Sea A una fórmula bien formada y cerrada cualquiera de TRIO' de longitud finita. Entonces una interpretación adecuada dada:

1) satisface A en el instante i (S_i (A) = true) sii el algoritmo comenzando con (A,i,{},T) es exitoso.

2) no satisface A en el instante i (S_i (A) = false) sii el algoritmo comenzando con (A,i,{},F) es exitoso.

3) Además, si el nodo (F, i, asg, V) no está marcado, F tiene una variable infinita x libre[XIX] o i tiene una variable infinita, entonces asg indica los valores que toma dicha variable para que se cumpla el valor de verdad que tiene el nodo.

Demostración:

Como se dijo anteriormente se hará inducción sobre la estructura de una fórmula TRIO', además el caso base se prueba también por inducción pero sobre la cantidad de variables TD.

Sea (A,i,{},V) la raíz de un árbol, se demuestra que se cumple 1) , 2) y 3) :

Caso Base sobre la estructura de la fórmula (A es una fórmula atómica cerrada):

Si A es una fórmula atómica, por ser cerrada es ground o solo tiene variables TD.

✧ *Caso base: A es una fórmula atómica* $p(v_1,.....,v_n)$, *ground,*

- *La construcción del árbol no hace nada.*

[XIX] Que esté libre, no solo indica que F está dentro del alcance de un ∀x, por solicitar el algoritmo fórmulas cerradas, sino que además es la única variable infinita ya que no se permiten variables infinitas anidadas Ver lema 1.

- La parte de evaluación hace:

· *Si p es un predicado TI entonces:*

✧ *Si V=T*

La tupla de valores $(v_1,.....,v_n) \notin \Pi(p)$ *sii el nodo es marcado. Por lo tanto*

S_i *(A) = true sii la tupla de valores* $(v_1,.....,v_n) \in \Pi(p)$ *sii el nodo no es marcado sii el algoritmo es exitoso.*

✧ *Si V=F*

La tupla de valores $(v_1,.....,v_n) \in \Pi(p)$ *sii el nodo es marcado. Por lo tanto*

S_i *(A) = false sii la tupla de valores* $(v_1,.....,v_n) \notin \Pi(p)$ *sii el nodo no es marcado sii el algoritmo es exitoso.*

· *Si p es un predicado TD entonces:*

Si $i \in T_f$

✧ *Si V=T*

S_i *(A) = true sii la tupla de valores* $(v_1,.....,v_n) \in \Pi_i(p)$ *sii el nodo no es marcado sii el algoritmo es exitoso.*

✧ *Si V=F*

S_i *(A) = false sii la tupla de valores* $(v_1,.....,v_n) \notin \Pi_i(p)$ *sii el nodo no es marcado sii el algoritmo es exitoso.*

Si $i \notin T_f$

• *Si V=T*

S_i *(A) = true sii la tupla de valores* $(v_1,.....,v_n) \in \Pi_e(p)$ *sii el nodo no es marcado sii el algoritmo es exitoso.*

• *Si V=F*

S_i *(A) = false sii la tupla de valores* $(v_1,.....,v_n) \notin \Pi_e(p)$ *sii el nodo no es marcado sii el algoritmo es exitoso.*

Ya que no hay variables infinitas, vale para cualquier valor de asg, en particular para el asg obtenido, por lo que vale 3) para este caso.

✧ *Caso inductivo: A es una fórmula atómica con n variables TD.*

Supongamos que z es la n-esima variable TD y que vale para las n-1 variables TD restantes.

- La construcción del árbol reemplaza la variable z por $\eta_i(z)$*, supongamos que* $\eta_i(z) = a$.

- La parte de evaluación hace:

✧ *Si V=T : El algoritmo es exitoso sii el nodo no está marcado sii por hipótesis inductiva* S_i ($A(^{z}/_{a})$) = *true sii* $S_i(A)$=*true para* $\eta_i(z)=a$.

✧ *Si V=F : El algoritmo es exitoso sii el nodo no está marcado sii por hipótesis inductiva* S_i ($F(^{z}/_{a})$) = *false sii* $S_i(A)$=*false para* $\eta_i(z)=a$.

Igual que en el caso anterior el algoritmo no asigna nada a asg, pero como no tiene variable infinita, entonces vale 3) para cualquier asg.

Por lo que podemos concluir que vale 1) , 2) y 3) para fórmulas atómicas cerradas.

Caso inductivo sobre la estructura de la fórmula:

- *A es de la forma ¬B*

  V=T

  - La construcción de árbol crea un nodo hijo con (B,i,asg,F).

  - La parte de evaluación hace:

  • Si no es exitoso el algoritmo, entonces el nodo hijo está marcado. Por lo tanto, si el nodo con (B,i,asg,F) está marcado, por hipótesis inductiva no se cumple que $S_i(B)$ = false por lo tanto $S_i(B)$ = true con lo que $S_i(\neg B)$ = false, con lo que $S_i(A)$ = false.

  • En caso contrario el algoritmo es exitoso. Entonces como el nodo hijo no estuvo marcado para (B,i,asg,F), por hipótesis inductiva se cumple que $S_i(B)$ = false por lo tanto $S_i(\neg B)$ = true con lo que $S_i(A)$ = true.(i)

  V=F

  - La construcción de árbol crea un nodo hijo con (B,i,asg, T).

  - La parte de evaluación hace:

  • Si el algoritmo no es exitoso, el nodo hijo está marcado. Por lo tanto si el nodo con (B,i,asg,T) está marcado, por hipótesis inductiva no se cumple que $S_i(B)$ = true por lo tanto $S_i(B)$ = false con lo que $S_i(\neg B)$ = true, con lo que $S_i(A)$ = true.

  • En caso contrario, el algoritmo es exitoso, entonces como no está marcado el nodo para (B,i,asg,T), por hipótesis inductiva se cumple que $S_i(B)$ = true por lo tanto $S_i(\neg B)$ = false, con lo que $S_i(A)$ = false.(ii)

  - Ya que (i) y (ii) no modifican los valores de las variables infinitas, y por hipótesis inductiva, asg indica los valores que toma la variable infinita libre para el nodo hijo, entonces vale 3) para el mismo asg, por lo tanto vale 3) para este caso.

Por consiguiente vale 1) , 2) y 3) para A=¬B.

- *Si $A=B \wedge C$:*

  - La primer parte del algoritmo crea dos nodos hijos con (B,i,asg,V) y con (C,i,asg,V) cada uno.

- La parte de evaluación, hace:

Si V=T

• Si el algoritmo es exitoso, entonces ninguno de sus hijos está marcado por lo tanto, por hipótesis inductiva S_i (B) = true y S_i (C) = true, con lo que $S_i(A)$=true.(iii)

• El algoritmo no es exitoso, entonces al menos uno de sus hijos está marcado, digamos el nodo que contiene la subfórmula B, entonces por hipótesis inductiva S_i (B) no es true con lo que S_i (A) no es true. Análogamente si es el nodo que contiene la subfórmula C. Por lo tanto $S_i(A)$no es true.

Si V=F

• Si el algoritmo es exitoso, entonces al menos uno de sus hijos no está marcado por lo tanto por hipótesis inductiva S_i (B) es false o S_i (C) es false, por lo tanto, $S_i(A)$ = false. (iv)

• El algoritmo no es exitoso, entonces todos sus hijos están marcados, entonces por hipótesis inductiva S_i (B) no es false ni S_i (C) es false con lo que S_i (A) no es false.

-En (iii) y (iv) por hipótesis inductiva asg indica los valores que toma la variable infinita libre o las que están en cada i, para cada nodo hijo, llamemos asg_B al asg obtenido en el nodo (B,i,asg,V) y análogamente asg_C para (C,i,asg,V), entonces:

Supongamos que ni la fórmula A contiene variables infinitas libres, ni i contiene variable infinita, por lo que ni B, ni C contienen variables infinitas libres, ni i de los nodos hijos, entonces no se necesitan valores de asg, o sea para asg no se asigna nada como el algoritmo dice.

En cada nodo hay una variable infinita que cumpla con esta condición, entonces por lema 2 es la misma, por lo que A vale : en (iii) para los mismos valores de la variable que están en los dos nodos, o sea para $asg_B \cap asg_C$ y en (iv) para cualquier valor de la variable que está en los dos nodos, o sea para $asg_B \cup asg_C$, o sea como el algoritmo dice.

Hay una variable infinita que cumpla la condición en un solo nodo, entonces A vale para los mismos valores de la variable que están en ese nodo, ya que no interesa el asg del nodo hermano, pues no tiene variable infinita libre (ni en i obviamente), o sea como el algoritmo dice.

Por lo tanto vale 1), 2) y 3) para este caso.

Si A = $\forall x$ B :

• x no es la variable infinita:

- La primer parte del algoritmo crea | D(x) | nodos hijos, cada uno con $(B^x/_a, i, asg, V)$, $\forall a \in D(x)$.

- La parte de evaluación hace:

Si V=T

• *Si el algoritmo es exitoso, entonces ninguno de sus nodos hijos está marcado, por lo tanto, por hipótesis inductiva* $S_i(B^x/_a)$ *es true para todo* $a \in D(x)$, *con lo que* $S_i(\forall x\, B)$=*true, por consiguiente* $S_i(A)$=*True. (v)*

• *El algoritmo no es exitoso, por lo tanto alguno de sus hijos está marcado, digamos el nodo que contiene la tupla* $(B^x/_{a1}, i, asg, T)$, *entonces por hipótesis inductiva* $S_i(B^x/_{a1})$ *no es true, con lo que* $S_i(\forall x B)$ *no es true, como se tomó un nodo hijo arbitrariamente, entonces vale para cualquier nodo hijo marcado, por consiguiente* $S_i(A)$ *no es true.*

Si V=F

• *El algoritmo es exitoso, entonces al menos uno de sus nodos hijos no está marcado, digamos el que asigna a x el valor a1, entonces por hipótesis inductiva* $S_i(B^x/_{a1})$ *es false para* $a1 \in D(x)$, *por lo tanto,* $S_i(\forall x B)$ = *false, por consiguiente* $S_i(A)$ = *false. (vi)*

• *El algoritmo no es exitoso, entonces, todos sus nodos hijos están marcados, entonces para todo* $a_i \in D(x)$, *el nodo que contiene la tupla* $(B^x/_{ai}, i, asg, F)$ *está marcado, entonces por hipótesis inductiva* $S_i(B^x/_{a1})$ *no es false para ningún* a_i, *con lo que* $S_i(\forall x B)$ *no es false, por consiguiente* $S_i(A)$ *no es false.*

- Supongamos que ni la fórmula A contiene variables infinitas libres, ni i contiene variables infinitas, entonces no se necesitan valores de asg, o sea para asg no se asigna nada como el algoritmo dice.

Supongamos, ahora el caso contrario, en (v) y (vi) por hipótesis inductiva, el conjunto asg de cada nodo hijo indica los valores que toman las variables infinitas libres del mismo entonces por lema 3, los nodos hijos tienen la misma y única variable infinita libre o en i, por lo que, entonces para el nodo padre los valores de la variable infinita son : en (v) los mismos valores de la variable infinita que están en todos los nodos hijos, o sea para la intersección de los mismos y en (vi) para cualquier valor de la variable infinita que está en los nodos hijos válidos, o sea para la unión de los mismos que no están marcados, o sea como el algoritmo dice.

• *x es la variable infinita:*

- La primer parte del algoritmo crea un nodo hijo con (B, i, asg, V).[xx]

- La parte de evaluación hace:

Si V=T

• *Si el algoritmo es exitoso, entonces su nodo hijo no está marcado y asg del mismo es igual a D(x), por lo tanto, por hipótesis inductiva* $S_i(B)$ *es true para todo valor de la variable infinita que está en asg, que por Corolario 1 es x, o sea para todo valor de x, con lo que* $S_i(\forall x\, B)$=*true, por consiguiente* $S_i(A)$=*True.*

• *El algoritmo no es exitoso, por lo tanto, o su hijo está marcado o asg del mismo es distinto de D(x), entonces por hipótesis inductiva, o* $S_i(B)$ *no es true o* $S_i(B^x/_a)$ = *false tal que* $\exists\, a \in D(x)$, $a \notin asg$, *con lo que* $S_i(\forall x B)$ *no es true, por consiguiente* $S_i(A)$ *no es true.*

Si V=F

[xx] Notar que en B , x está libre.

• *Si el algoritmo es exitoso, entonces su nodo hijo no está marcado y asg del mismo es distinto del conjunto vacío, por lo tanto, por hipótesis inductiva S_i (B) es false para algún valor de la variable infinita, que por Corolario 1 es x, que $\in$ a D(x), con lo que $S_i(\forall x\ B)$=false, por consiguiente $S_i(A)$=false para el mismo asg.*

• *El algoritmo no es exitoso, por lo tanto, o su hijo está marcado o asg del mismo es igual a vacío, entonces por hipótesis inductiva, o S_i (B) no es false o $S_i(B^x/_a)$= true, $\forall a \in D(x)$, $a \notin asg$, con lo que S_i ($\forall x B$) no es false, por consiguiente S_i (A) no es false.*

Por lo tanto vale 1) , 2) y 3) para este caso.

➢*Si A= Dist(B,t):*

• *t no tiene ninguna variable:*

⌑ *Si | asg | ≤ 1*

- *La primer parte del algoritmo obtiene k como i+$S_i(t)$ y crea un nodo hijo con (B, k , asg, V).*

- *En la parte de evaluación:*

⌖ *V=T*

• *El algoritmo es exitoso, entonces el nodo hijo no está marcado, entonces por hipótesis inductiva $S_k(B)$ es true para k=i+$S_i(t)$, con lo que S_i(Dist(B,t)) es true, por lo tanto $S_i(A)$ es true.(vii)*

• *El algoritmo no es exitoso, el nodo hijo está marcado, entonces por hipótesis inductiva $S_k(B)$ no es true para k=i+$S_i(t)$, por lo tanto S_i(Dist(B,t)) no es true, con lo que $S_i(A)$ no es true.*

⌖ *V = F*

• *El algoritmo es exitoso, el nodo hijo no está marcado, por lo tanto, por hipótesis inductiva $S_k(B)$ es false para k=i+$S_i(t)$, con lo que S_i(Dist(B,t)) es false, por lo tanto Si(A) es false.(viii)*

• *El algoritmo no es exitoso, por lo que el nodo hijo está marcado, entonces por hipótesis inductiva $S_k(B)$ no es false para k=i+$S_i(t)$, por lo tanto S_i(Dist(B,t)) no es false, con lo que $S_i(A)$ no es false.*

Ya que (vii) y (vii) no modifican los valores de las variables infinitas, y por hipótesis inductiva, asg indica los valores que toma la variable infinita libre para el nodo hijo, entonces vale 3) para el mismo asg, por lo tanto vale 3) para este caso.

⌑ *Si | asg | >1, entonces, i representa una expresión donde la variable infinita toma más de un valor, que están indicados en asg, con lo que puede haber nuevos valores para que el instante de tiempo esté dentro de T_f, por lo que el algoritmo vuelve a calcular los valores de la variable infinita.*

- *La primer parte del algoritmo hace, si x es la variable infinita que está en i:*

temp = { x' / $S_i(t)+i^x/_{x'} \in T_f$, $x' \in asg$ }

temp'= asg - temp

Si temp $\neq$ {} : Crea | temp | nodos hijos, con (B, $S_i(t)+i^x/_{x'}$, {x'} ,V), $\forall x' \in temp$.

Crea un nodo hijo con $(B, S_i(t)+i, temp', V)$.

Es decir si temp = {}, entonces no se volvió a entrar a T_f. *Temp' no puede ser vacío ya que, igual que asg, es la diferencia entre un conjunto infinito y uno finito.*

- En la parte de evaluación:

Φ $V=T$

• *El algoritmo es exitoso, al menos un nodo hijo no está marcado, entonces por hipótesis inductiva* $S_k(B)$ *es true, para* $k= S_i(t)+i^x/_{x'}$, $x' \in asg$, *con lo que* $S_i(Dist(B,t))$ *es true para todos los asg de los hijos sin marcar, por lo tanto* $S_i(A)$ = *true, para la unión de los asg de los hijos sin marcar.*

• *El algoritmo no es exitoso, por lo tanto todos los nodos hijos están marcados, entonces por hipótesis inductiva* $S_k(B)$ *no es true para* $k= S_i(t)+i^x/_{x'}$, $x' \in asg$, *por lo tanto* $S_i(Dist(B,t))$ *no es true para los valores del asg de los hijos, con lo que* $S_i(A)$ *no es true para el asg indicado, que por hipótesis inductiva es el valor que toma la variable infinita, por lo tanto no* $\exists S_i$ *tal que* $S_i(A)$ *es True.*

Φ $V = F$

• *El algoritmo es exitoso, entonces al menos un nodo hijo no está marcado, entonces por hipótesis inductiva* $S_k(B)$ *es false, para* $k= S_i(t)+i^x/_{x'}$, $x' \in asg$, *con lo que* $S_i(Dist(B,t))$ *es false para todos los asg de los hijos sin marcar, por lo tanto* $S_i(A)$ = *false, para la unión de los asg de los hijos sin marcar.*

• *El algoritmo no es exitoso, por lo tanto todos los nodos hijos están marcados, entonces por hipótesis inductiva* $S_k(B)$ *no es false para* $k= S_i(t)+i^x/_{x'}$, $x' \in asg$, *por lo tanto* $S_i(Dist(B,t))$ *no es false para los valores del asg de los hijos, con lo que* $S_i(A)$ *no es false para el asg indicado, que por hipótesis inductiva es el valor que toma la variable infinita, por lo tanto no* $\exists S_i$ *tal que* $S_i(A)$ *es True.*

• *Si t contiene solo variables TI y valores ground, entonces crea un nodo hijo con* $(A, i+t, asg, V)$[xxi], *entonces*

Supongamos que x es la variable infinita que está en t,

- La primer parte del algoritmo hace:

$temp = \{ x' / i+S_i(t^x/_{x'}) \in T_f, x' \in D(x) \}$

$temp' = D(x) - temp$

Crea $|temp|$ *nodos hijos, con* $(B, i+S_i(t^x/_{x'}), \{x'\}, V)$, $\forall x' \in temp$.

Crea un nodo hijo con $(B, i+t, temp', V)$.

- En la parte de evaluación:

Φ $V=T$

[xxi] Notar que, como dijimos en el algoritmo, por ser fórmulas cerradas y por la construcción del árbol ,la única variable que puede estar no instanciada a esta altura de algoritmo en el término t, es una variable infinita.

• El algoritmo es exitoso, entonces al menos un nodo hijo no está marcado, entonces por hipótesis inductiva $S_k(B)$, $k=i+S_i(t^x/_b)$[xxii], es true para b está en asg, que por construcción $\in D(x)$, o sea para el valor b de la variable infinita, con lo que $S_i(Dist(B,t))$ es true para el mismo asg y para los asg de los nodos hijos sin marcar, por lo tanto $S_i(A)$ es true para la unión de los asg de los nodos hijos sin marcar, o sea el asg indicado.

• El algoritmo no es exitoso, por lo que todos los nodos hijos están marcados, entonces por hipótesis inductiva $S_k(B)$, $k=i+ S_i(t^x/_b)$ no es true para ningún b que está en asg de ningún nodo, o sea para ningún $b \in D(x)$, por lo tanto $S_i(Dist(B,t))$ no es true para cualquier valor de la variable infinita, con lo que $S_i(A)$ no es true.

$\Phi V = F$

• El algoritmo es exitoso, al menos un nodo hijo no está marcado, entonces por hipótesis inductiva $S_k(B)$, $k=i+ S_i(t^x/_b)$, es false para b está en asg, que por construcción $\in D(x)$, o sea para el valor b de la variable infinita, con lo que $S_i(Dist(B,t))$ es false para el mismo asg y para los asg de los nodos hijos sin marcar, por lo tanto $S_i(A)$ es false para la unión de los asg de los nodos hijos sin marcar, o sea el asg indicado.

• El algoritmo no es exitoso, por lo que todos los nodos hijos están marcados, entonces por hipótesis inductiva $S_k(B)$ no es false, $k=i+ S_i(t^x/_b)$ para ningún b que está en asg de ningún nodo, o sea para ningún $b \in D(x)$, por lo tanto $S_i(Dist(B,t))$ no es false para cualquier valor de la variable infinita, con lo que $S_i(A)$ no es false.

Por lo tanto vale 1) , 2) y 3) para este caso.

Por lo que queda demostrado.

[xxii] Notar que $S_i(t^x/_b) = t^x/_b$ ya que t solo contiene valores ground.

Bibliografía

[Aba87] Abadi, M. ' Temporal Logic.Theorem Proving. PhD Thesis, Stanford University, 1987.

[ADC90] Alur, R. Courcoubetis, C. , Dill, D. 1990 .'Model-checking for real-time system', Proc. 5th. Symp. on Logics in Computer Science. IEEE Computer Society Press.

[BPM83] Ben-Ari, M., Pnueli, A., Manna, Z. 1977. 'The temporal logic of branching time'.Acta Inf., (1983), pp 207-226.

[CCMSP93] Ciapessoni E.,Corsetti E., Montanari A., San Pietro P. , 'Embedding time granulaeity in a logical specification languaje for synchromous real-time systems' . Science of Computer Programming 20. Elsevier Science Publishers. 1993.

[CE81] Clarke, E., M., Emerson, A. E. ' Design and synthesis of synchronization skeletons using branching time temporal logic '. IBM Logics of Programs Workshop. Lectures Notes in Computer Science, vol.131, Springer-Verlag, N.Y. , 1981.

[EH86] Emerson, A., Halpern, J.. . ' "Sometimes" and "Not Never" Revisited: On Bramching versus Linear Time Temporal Logic'. Journal of ACM, Vol 33. Nro. 1 January 1986. pp 151-178.

[FM94] Felder, M., Morzenti A., 'Validating real-time systems by executing logic specifications in TRIO ' . ACM Transactions on Software Engineeering and Methodologies, Vol. 3 No. 4, October 1994 pp 308-339.

[FSP92] Felder, M. San Pietro, P., ' Finite Time semantics for executable logic specifications'. NATO ASI on Real-Time Computing, Sanit Martin, Dutch Antille, October 1992. Springer Verlag.

[GMM89] Ghezzi, C., Mandrioli, D., Morzenti, A.,1989. 'TRIO, a logic language for executable specification of real-time system'. Proceedings of 10th French-Tunisian Seminar on Computer Science. Tunis. pp 322-349.

[GMMP91] Ghezzi, C., Mandrioli, D., Morzenti, A.,Pezzé, M., 1991. ' A unified high-level Petri net formalism for time-critical systems'. IEEE Transactions on Software Engineering, Vol.17, No. 2.

[GPSS80] Gabbay, D., Pnueli, A., Shela, S., Stavi, J. 1980. ' On the temporal analisys of fairness'. Proceedings of the 12th. Annual ACM Symposium on Principles of Programming Languages (New Orleans). ACM, New York.

[HC68] Hughes G.E. and Cresswell M.J. 'An Introduction to Modal Logic', Methuen, N.Y. [1968]

[HNSY94] Henzinger, T. A., Nicollin, X., Sifakis, J., Yovine, S. 1994. 'Symbolic Model Cheking for Real-Time System'.Information and Computation, Academic Press, Belgica.

[La83] Lamport, L. , 1983. 'Specifyng Concurrent Program Modules.' ACM Transactions on Prog. Lang and Sys.

[LPZ83] Lichtenstein O., Pnueli A., Zuck L.D.,1985. 'The Glory of the Past'. Logics of Programs, Lecture Notes in Computer Science 193, Springer-Verlag.

[Men79] Mendelson, E. 1979. ' Introduction to Mathematical Logic'. Van Nostrand Company, New York.

[MMG92]Morzenti, A., Mandrioli, D., Ghezzi, C.,1992 . ' A Model Parametric Real Time Logic', ACM Trans.,on Programming languages and Systems, Vol 14, Nro. 4, October 1992, pp 521-573.

[Mor89] Morzenti, A. 1989. 'The Specification of Real-Time Systems: Proposal of a Logic Formalism', PhD Thesis, Dipartamento di Elettronica, Politecnico di Milano.

[MP81] Manna, Z , Pnuelli, A. 1981. ' Verification of Concurrent Programs: The Temporal Framework.' Proc. Workshop on Logics of Programs, D.C.Kozan, Lec. Notes in Comp. Sci. 131,Springer-Verlag.

[MSP91] Morzenti, A , San Pietro P. 1991. ' An object-oriented language for modular system specifications' P America,ed., Proceedings ECOOP'91 . Springer-Verlag .

[Pn77] Pnueli, A. 1977. 'The temporal semantics of concurrent programs'. 18th Symp. Found. Comput. Sci.,Providence, pp 46-57.

[QS81] Quille, J. P., Sifakis, J. 1981. ' Specification and Verification of Concurrent System in Cesar' . Proc. Fifth Int. Symp. Program . Springer Lect. Notes Comput. Sci. 137:337-si.

[Sm68] Smullian, R. M. 1968. ' First Order Logic ' . Springer-Verlag, New York.

[Wo83] Wolper, p. 1977. 'Temporal logic can be more expressive'.Inf. Control 56 (1983), pp 72-99.

Trabajo de Grado

Anexo

TRIO': O como ahora las tautologías son tautologías y el infinito es infinito.

Isaac Carlos Turquie.

ÍNDICE

Prototipo del algoritmo de Generación de Modelos

En este capítulo se detallan los programas fuentes del algoritmo de Generación de Modelos.

PROGRAMA PRINCIPAL GEN-MOD.

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$M 54817,8192}
{ Programa de Generación de Modelos. (c) Isaac Carlos Turquie }

PROGRAM gen_mod;

{ Evalua una fórmula dada con la interpretacion de las unit corresp.
  ATENCION: Las formulas se especificaran sin blancos y con los siguientes
    simbolos, utilizados como palabras reservadas, no pudiendo ser usados
    como nombres de variables, funcion o predicados:
                # : cuantificador universal,
                & : And,
                - : Not,
                % : Dist,
                () Separador
            Ademas la variable, al lado del cuantificador no lleva ()
    Los Predicados son de una letra.            }

Uses construc,evaluar,arboles,wincrt;

var
   A1:arbol;
   punt:parb;
   form,i:string;
   exito:boolean;

begin
   WindowTitle[0]:=' ';
```

```
WindowTitle[1]:='A';
WindowTitle[2]:='l';
WindowTitle[3]:='g';
WindowTitle[4]:='o';
WindowTitle[5]:='r';
WindowTitle[6]:='i';
WindowTitle[7]:='t';
WindowTitle[8]:='m';
WindowTitle[9]:='o';
WindowTitle[10]:=' ';
WindowTitle[11]:='d';
WindowTitle[12]:='e';
WindowTitle[13]:=' ';
WindowTitle[14]:='G';
WindowTitle[15]:='e';
WindowTitle[16]:='n';
WindowTitle[17]:='e';
WindowTitle[18]:='r';
WindowTitle[19]:='a';
WindowTitle[20]:='c';
WindowTitle[21]:='i';
WindowTitle[22]:='ó';
WindowTitle[23]:='n';
WindowTitle[24]:=' ';
WindowTitle[25]:='d';
WindowTitle[26]:='e';
WindowTitle[27]:=' ';
WindowTitle[28]:='M';
WindowTitle[29]:='o';
WindowTitle[30]:='d';
WindowTitle[31]:='e';
WindowTitle[32]:='l';
WindowTitle[33]:='o';
WindowTitle[34]:='s';
WindowTitle[35]:='.';
windowOrg.x:=0;
windowOrg.y:=0;
```

```
windowSize.x:=700;
windowSize.y:=500;
ScreenSize.x:=100;
ScreenSize.y:=600;
InitWinCrt;
clrscr;
writeln;
writeln(' Las fórmulas se especificaran sin blancos y con los siguientes');
writeln('símbolos, utilizados como palabras reservadas, no pudiendo ser usados');
writeln('como nombres de variables o predicados:  ');
writeln('          # : cuantificador universal,');
writeln('          & : And,                  ');
writeln('          - : Not,                 ');
writeln('          % : Dist,                 ');
writeln('          () Separador              ');
writeln('       Además la variable, al lado del cuantificador no lleva () ');
writeln;
Writeln('Ingrese una Fórmula para evaluar (ENTER para terminar) : ');
readln(form);
while (form<>'') do
begin
   writeln;
   write('Ingrese el instante de tiempo en el que se va a evaluar : ');
   readln(i);
   pri:=nil;
   A1:=crear(form,i);
   AgregarArbol(A1);
   Procesar(A1,A1^.raiz);
   punt:=pri;
   exito:=false;
   while punt<>nil do
      begin
         Marcar(punt^.nodo^.raiz);
         verArbol(punt^.nodo);
         writeln(' ------------------------------------------------------------------------');
         if (punt^.nodo^.raiz^.marcado)
```

```
                then writeln(' No se satisface la fórmula en el instante '+i+' para las asignaciones
indicadas.')
                 else exito:=true;
               writeln;
               writeln('Presione cualquier tecla');
               readkey;
               punt:=punt^.sig;
            end;
         writeln;
         if exito
            then begin
               writeln('                              È X I T O');
               writeln('        Existe una interpretación, a partir del frame dado,');
               writeln('            que satisface la fórmula en el instante '+i+'.');
               end
            else begin
               writeln('        NO Existe una interpretación, a partir del frame dado,');
               writeln('            que satisface la fórmula en el instante '+i+'.');
               end;
         writeln;
         writeln('Presione cualquier tecla');
         readkey;
         clrscr;
         writeln;
         Writeln('Ingrese una Fórmula para evaluar (ENTER para terminar) : ');
         readln(form);
      end;
      donewincrt

end.
```

UNIDAD CONSTRUC

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}

UNIT Construc;
{ Construye todos los árboles de subfórmulas, por lo tanto,
 también evalua las fórmulas atómicas.}


INTERFACE

Uses terminos,arboles,frame,wincrt,conjunto;

Procedure Procesar(Arb:arbol;nod:ptero);
{ Dado el nodo nod del árbol arbprocesa este nodo según el algoritmo
de generación de modelos.}


IMPLEMENTATION


Procedure Procesar(Arb:arbol;nod:ptero);
 var
    h1:ptero;
    form2,taux,t1,t2: string;
    hij:nodos;

Procedure ProcFAtomica(n1:ptero);
var
    instante:integer;
    varTD1,varl:char;
    valorTD,cual,nocual,value:integer;
    re,valorstr,valorl,form,t1,f2:string;
    auxbool:boolean;
    h1,nod1:ptero;
    arb1:arbol;

begin
    with n1^ do
```

```
        begin
{---------- Construcción ----------}
          if HayVarInf(i,varl)
             then begin
                str(UnValor(asg),valorl);
                i:=replace(i,varl,valorl);
                end;
          terminoFormula(formula,form,t1);
          if HayVarTD(t1,varTD1)
             then begin
                instante:=EvaluarTermino(i);
                if HayAsgTD(Arb,varTD1,instante,valorTD)
                   then begin
                      str(valorTD,re);
                      F2:=form+'('+replace(t1,varTD1,re)+')';
                      h1:=newnodo(F2,i,asg,V);
                      newhijo(n1,h1);
                      Procesar(Arb,h1);
                      end
                   else begin
                      for valorTD:=min(varTD1) to max(varTD1)-1 do
                         begin
                         str(valorTD,re);
                         F2:=form+'('+replace(t1,varTD1,re)+')';
                         CopiarArbol(Arb,Arb1,n1,nod1);
                         AgregarArbol(Arb1);
                         AgregarAsgTD(Arb1,varTD1,valorTD,instante);
                         h1:=newnodo(F2,i,asg,V);
                         newhijo(nod1,h1);
                         Procesar(Arb1,Arb1^.raiz);
                         end;
                      str(max(varTD1),re);
                      F2:=form+'('+replace(t1,varTD1,re)+')';
                      AgregarAsgTD(Arb,varTD1,max(varTD1),instante);
                      h1:=newnodo(F2,i,asg,V);
                      newhijo(n1,h1);
                      Procesar(Arb,h1);
                      end
                end
          else begin
{---------- Marcado ----------}
```

```
if predicadoTI(form)
   then if (V and (Not P(formula)))or((Not V) and P(formula) )
      then marcado:=true
      else marcado:=false
   else begin
      instante:=EvaluarTermino(i);
      if V
         then begin
            cual:=1;
            nocual:=2;
            end
         else begin
            cual:=2;
            nocual:=1;
            end;
      value:=evaluarTermino(t1);
      if ((instante<=maxT) and (instante>=minT))
         then begin
            if Not(AgregarPI(cual,Arb,n1,form,instante,value))
               then begin
                  marcado:=true;
                  if not (FueSacadodePI(cual,form,instante,value))
                     then begin
                        copiarArbol(Arb,Arb1,n1,nod1);
                        AgregarArbol(Arb1);
                        SacarPI(nocual,Arb,form,instante,value);
                        auxbool:=AgregarPi(cual,Arb,n1,form,instante,value);
                        Procesar(Arb1,Arb1^.raiz);
                        marcado:=false;
                        end;
                   end
               else marcado:=false;
            end
         else begin
            if Not(AgregarPe(cual,Arb,n1,form,value))
               then begin
                  marcado:=true;
                  if not (FueSacadoDePe(cual,form,value))
                     then begin
                        copiarArbol(Arb,Arb1,n1,nod1);
                        AgregarArbol(Arb1);
```

```
                                SacarPe(nocual,Arb,form,value);
                                auxbool:=AgregarPe(cual,Arb,n1,form,value);
                                Procesar(Arb1,Arb1^.raiz);
                                marcado:=false;
                                end;
                            end
                        else marcado:=false;
                    end;
                end;
            end;
        end;
end;


Procedure ProcAnd(F1,F2:string);
var h1,h2:ptero;
begin
    with nod^ do
        begin
            h1:=newnodo(F1,i,asg,V);
            newhijo(nod,h1);
            h2:=newnodo(F2,i,asg,V);
            newhijo(nod,h2);
            Procesar(Arb,h1);
            Procesar(Arb,h2);
        end;
end;

Procedure ProcPt(n1:ptero);
 var form2,t1,valorstr:string;
    indice,k:integer;
    hij:nodos;
    h1:ptero;
    aux:char;
begin
    with n1^ do
    begin
        If formula[3]='('
            then t1:=copy(formula,4,length(formula)-4)
            else t1:=copy(formula,3,255);
        if (formula[2]In varInfinitas)
```

```
        then
            begin
                h1:=newnodo(t1,i,asg,v);
                newhijo(n1,h1);
                Procesar(arb,h1);
             end
        else begin
            for indice:=min(formula[2]) to max(formula[2]) do
                begin
                    str(indice,valorstr);
                    form2:=replace(t1,formula[2],valorstr);
                    h1:=newnodo(form2,i,asg,V);
                    newhijo(n1,h1);
                end;
            hij:=hijos;
            while hij<>nil do
                begin
                    Procesar(arb,hij^.nodo);
                    hij:=hij^.sig;
                end;
            end;
    end;
end;


Procedure ProcDist(n1:ptero);
 var instante,value,minTemp,maxTemp:integer;
    varTD1,varl2,varl:char;
    valorTD:integer;
    asg1,strl,strTD,j,inTemp,form,t1,t2,outTemp,re,newInst:string;
    hij:nodos;
    aux:char;
    h1,nod1:ptero;
    arb1:arbol;
 begin
    with n1^ do
    begin
        {----- Obtengo Si(T2)----}
        if HayVarInf(i,varl)
            then begin
                str(UnValor(asg),strl);
```

```
        j:=replace(i,varl,strl)
        end
    else j:=i;
tomartermDist(formula,t1,t2);
instante:=EvaluarTermino(j);
if HayVarTD(t2,varTD1)
   then  if HayAsgTD(Arb,varTD1,instante,valorTD)
      then begin
         str(valorTD,re);
         form:='%('+t1+','+replace(t2,varTD1,re)+')';
         h1:=newnodo(form,i,asg,V);
         newhijo(n1,h1);
         Procesar(Arb,h1);
         end
      else begin
         for valorTD:=min(varTD1)+1 to max(varTD1) do
            begin
               str(valorTD,re);
               form:='%('+t1+','+replace(t2,varTD1,re)+')';
               CopiarArbol(Arb,Arb1,n1,nod1);
               AgregarArbol(Arb1);
               AgregarAsgTD(Arb1,varTD1,valorTD,instante);
               h1:=newnodo(form,i,asg,V);
               newhijo(nod1,h1);
               Procesar(Arb1,Arb1^.raiz);
            end;
         str(min(varTD1),re);
         form:='%('+t1+','+replace(t2,varTD1,re)+')';
         AgregarAsgTD(Arb,varTD1,min(varTD1),instante);
         h1:=newnodo(form,i,asg,V);
         newhijo(n1,h1);
         Procesar(Arb,h1);
      end
else begin { if hayvarTD....}
   if HayVarInf(t2,varl2)
      then begin
         if t2[1]='-'
            then  t2:=i+t2
            else  t2:=i+'+'+t2;
         ObtenerTemp(t2,Dominio(varl2),inTemp,outTemp);
         while HayySacarIntervalo(inTemp,varl2,minTemp,maxTemp)do
```

```
            { InTemp solo puede tener intervalos }
            for value:=minTemp+1 to maxTemp-1 do
                begin
                    str(value,re);
                    NewInst:=replace(t2,varI2,re);
                    asg1:='';
                    AgregarUnValor(asg1,varI2,value);
                    h1:=newnodo(t1,NewInst,asg1,V);
                    newhijo(n1,h1);
                end;
        h1:=newnodo(t1,t2,OutTemp,V);
        newhijo(n1,h1);
        end
else If MasdeUnElem(asg)
        then begin
            if t2[1]='-'
                then  t2:=i+t2
                else  t2:=i+'+'+t2;
            ObtenerTemp(t2,asg,inTemp,outTemp);
            while HayySacarIntervalo(inTemp,varI,minTemp,maxTemp)do
                for value:=minTemp+1 to maxTemp-1 do
                    begin
                    str(value,re);
                    NewInst:=replace(t2,varI,re);
                    asg1:='';
                    AgregarUnValor(asg1,varI,value);
                    h1:=newnodo(t1,NewInst,asg1,V);
                    newhijo(n1,h1);
                    end;
            h1:=newnodo(t1,t2,OutTemp,V);
            newhijo(n1,h1);
            end
        else begin
            if t2[1]='-'
                then  t2:=j+t2
                else  t2:=j+'+'+t2;
            h1:=newnodo(t1,t2,asg,V);
            newhijo(n1,h1);
            end;
hij:=hijos;
while hij<>nil do
```

```
            begin
                Procesar(arb,hij^.nodo);
                hij:=hij^.sig;
            end;
        end; { hay varTD....}
    end;
end;


begin  { de Procesar }

    if (nod^.procesado)
        then begin
            hij:=nod^.hijos;
            while (hij<>nil) do
                begin
                    Procesar(Arb,hij^.nodo);
                    hij:=hij^.sig;
                end;
            end
        else begin
            nod^.procesado:=true;
            case nod^.formula[1] of
                '-':with nod^ do
                    begin
                        form2:=copy(formula,2,255);
                        tomartermino(form2,taux);
                        if length(taux)<length(form2)
                            then begin              { 2 términos y el & }
                                t1:='-'+taux;
                                t2:=copy(formula,length(t1)+2,255);
                                ProcAnd(t1,t2);
                                end
                            else begin              { 1 término y el - }
                                t1:=copy(formula,2,255);
                                h1:=newnodo(t1,i,asg,not(V));
                                newhijo(nod,h1);
                                Procesar(Arb,h1);
                                end;
                    end;
                '#':with nod^ do
```

```
    begin
      form2:=copy(formula,3,length(formula)-2);
      tomartermino(form2,taux);
      if length(taux)<length(form2)
        then begin              { 2 términos y el & }
          t1:=formula[1]+formula[2]+taux;
          t2:=copy(formula,length(t1)+2,255);
          ProcAnd(t1,t2);
          end
        else ProcPT(nod);       { 1 término y el # }
    end;
  '(':with nod^ do
    begin
      tomartermino(formula,taux);
      if length(taux)<length(formula)
        then begin              { 2 términos y el & }
          t1:=taux;
          t2:=copy(formula,length(t1)+2,255);
          ProcAnd(t1,t2);
         end
        else begin              { 1 término entre parentesis }
          formula:=copy(taux,2,(length(taux)-2));
          procesado:=false;
          Procesar(Arb,nod);
          end;
    end;
  '%':with nod^ do
    begin
      form2:=copy(formula,2,255);
      tomartermino(form2,taux);
      if length(taux)<length(form2)
        then begin              { 2 términos y el & }
          t1:='%'+taux;
          t2:=copy(formula,length(t1)+2,255);
          ProcAnd(t1,t2);
          end
        else ProcDist(nod);     { 1 término y el dist }
    end;
  else with nod^ do
      begin
        tomartermino(formula,taux);
```

```
                    if length(taux)<length(formula)
                        then begin              { 2 términos y el & }
                            t1:=taux;
                            t2:=copy(formula,length(t1)+2,255);
                            procAnd(t1,t2);
                            end
                        else                    { fórmula atómica }
                            ProcFAtomica(nod);
                end;
            end;
        end;
end;


end.
```

UNIDAD EVALUAR

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}

UNIT evaluar;
 { Dado un árbol, hace el proceso de evaluación}

INTERFACE

Uses frame,terminos,arboles,conjunto;

Function Marcar(n1:ptero):boolean;
{ Dado el nodo n1, marca el mismo y todos los que están entre él y
 las fórmulas atómicas. }

IMPLEMENTATION

Function Marcar(n1:ptero):boolean;
 var tieneInf,marcanod,marcahij,dis:boolean;
    hij:nodos;
    asgnod,f2,t2:string;
    aux:char;
 begin
    with n1^ do
    if hijos<>nil
        then if (hijos^.sig=nil)
            then begin  { 1 solo hijo}
                marcanod:=marcar(hijos^.nodo);
                if (not marcanod)
                    then if ((formula[1]='#')and(formula[2] In varInfinitas))
                        then if V          { Es variable infinita y no está marcado }
                            then marcanod:=(Pos(dominio(formula[2]),hijos^.nodo^.asg)=0)
                            else marcanod:=Vacio(hijos^.nodo^.asg)
                        else asg:=hijos^.nodo^.asg;
                end
            else begin
                dis:= false;
                if ((formula[1]='%'))
                    then begin
```

```
            f2:=copy(formula,2,255);
            tomartermino(f2,t2);
            if length(t2)=length(f2)
               then dis:= true;
            end;
         hij:=hijos;
         tieneInf:=(HayVarInf(i,aux));
         if ((Not(V))or dis)
            then begin
               marcanod:=true;
               asgnod:='xxx';
               while (hij<>nil) do {siempre hay al menos un hijo}
                  begin
                     marcahij:= Marcar(hij^.nodo);
                     marcanod:=marcanod and marcahij;
                     if (not marcahij)and
                        (dis or tieneInf or(HayvarInf(hij^.nodo^.formula,aux)))
                        then if asgnod='xxx'
                           then asgnod:=hij^.nodo^.asg
                           else asgnod:=Union(asgnod,hij^.nodo^.asg);
                     hij:=hij^.sig;
                  end;
               end
            else begin
               marcanod:=false;
               asgnod:='xxx';
               while (hij<>nil) do { marco todos asi lo veo }
                  begin
                     marcahij:= Marcar(hij^.nodo);
                     marcanod:=marcanod or marcahij;
                     if ((HayvarInf(hij^.nodo^.formula,aux))or(tieneInf))
                        then if asgnod<>'xxx'
                           then asgnod:=Interseccion(asgnod,hij^.nodo^.asg)
                           else asgnod:=hij^.nodo^.asg;
                     hij:=hij^.sig;
                  end;
               end;
            if ((not marcanod)and(asgnod<>'xxx'))
                  then asg:=asgnod;
            end
      else marcanod:=marcado;   { fórmula atómica }
```

```
    n1^.marcado:=marcanod;
    Marcar:= marcanod;
 end;

end.
```

UNIDAD ARBOLES

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}

UNIT arboles;

{ Contiene las estructuras de datos de los nodos y los árboles del algoritmo
 y todas las funciones para manejarlos. }

INTERFACE

Uses wincrt,frame;

type
   asigTD=^regasigtd;
   regasigTD = record
          variable:char;
          valor:integer;
          i:integer;
          sig:asigTD;
   end;
   ptero=^nodo;
   nodos=^regnodos;
   regnodos= record
        nodo:ptero;
        sig:nodos;
   end;
   nodo = record
          formula:string;
          i:string;
          V:boolean;
          asg:string;
          hijos:nodos;
          marcado:boolean;
          procesado:boolean;
      end;
   hojas=^reghoja;
   reghoja= record
          leafs:nodos;
```

```
            valor:integer;
            sig:hojas;
            end;

    arb = record
            asgTD :asigtd;
            Pi:Array [TPredTD,minT..maxT] of hojas;  { predicado , tiempo }
            Pi_:Array [TPredTD,minT..maxT] of hojas; { Pi' }
            Pe:Array [TPredTD] of hojas;
            Pe_:Array [TPredTD] of hojas; { Pe' }
            raiz:ptero;
    end;

    arbol = ^arb;

    parb=^regArb;
    regArb= record
        nodo:Arbol;
        sig:parb;
    end;

var
    pri:parb;

Procedure AgregarAsgTD(A:arbol;vrbl:char;vlr,i:integer);
{ Agrega al árbol A una asignacion de la variable vrbl en el instante i con el valor vlr.}

Function HayAsgTD(A:arbol;VTD:char;j:integer; Var Value:integer):boolean;
{ Se fija si hay alguna asignacion, en el árbol A, en el instante j, de la variableTD VTD.
  Si hay asignacion, entonces, devuelve el valor de la variable en value, si no, devuelve 0}

Function AgregarPi(cual:integer;A:arbol;n:ptero;p:string;i:integer;vlr:integer):boolean;
{ Agrega al 'Pi', según cual, del árbol A del predicado p en el instante i el valor value
 y deja guardado que nodo lo hizo (n).
    Los valores de cual son: 1:Pi, 2:Pi_( o sea Pi') }

Function AgregarPe(cual:integer;A:arbol;n:ptero;p:string;vlr:integer):boolean;
{ Agrega al 'Pe', según cual, del árbol A del predicado p,
  el valor value y deja guardado que nodo lo hizo (n).
    Los valores de cual son: 1:Pe, 2:Pe_ ( o sea Pe') }
```

```
Procedure SacarPI(cual:integer;A:arbol;p:string;i,vlr:integer);
{ Saca y marca todos los nodos de la asignacion especificada en cual, para el predicado A,
 en el instante i , con el valor vlr.
    Los valores de cual son: 1:Pi, 2:Pi_.}

Procedure SacarPe(cual:integer;A:arbol;p:string;vlr:integer);
{Saca y marca todos los nodos de la asignacion especificada en cual,
  para el predicado A,con el valor vlr.
    Los valores de cual son: 1:Pe, 2:Pe_.}

Function FueSacadoDePI(cual:integer;p:string;i:integer;vlr:integer):boolean;
{ Devuelve true si, según cual, al 'Pi',del árbol A del predicado p en el instante i
 del valor value se sacó alguna vez.
    Los valores de cual son: 1:Pi, 2:Pi_( o sea Pi') }

Function FueSacadoDePe(cual:integer;p:string;vlr:integer):boolean;
{ Análogo pero para Pe y Pe'.}

Function newnodo(form,j,asg:string;val:boolean):ptero;
{crea un nuevo nodo y devuelve su ptero}

Procedure newhijo(var n1,n2:ptero);
{ agrega un nodo hijo n2 al nodo n1}

Function crear(formula,i:string):arbol;
{ crea un árbol con asgtd vacio }

Procedure CopiarArbol(A1:arbol;Var A2:arbol;n1:ptero;Var n2:ptero);
{ copia el árbol A1 en A2 y devuelve en n2 el nodo que correspondia a n1}

Procedure AgregarArbol(A1:arbol);
{ Agrega el árbol A1 }

Procedure verArbol(A:arbol);
{ Muestra en pantalla todos los datos de un árbol }



IMPLEMENTATION

var
```

```
    k:integer;
    j:TPredTD;

{ Indican que ya fue sacado de algún árbol, y por lo tanto creado en otro,
 los predicados en los instantes y con los valores especificados}
    SacadoPi:Array [TPredTD,minT..maxT] of set of byte;  { predicado , tiempo }
    SacadoPi_:Array [TPredTD,minT..maxT] of set of byte; { Pi' }
    SacadoPe:Array [TPredTD] of set of byte;
    SacadoPe_:Array [TPredTD] of set of byte; { Pe' }

{---------------------------------- PROC. Y FUNCIONES AUXILIARES ----------------------}

 Function IgualTiempo(t1,t2:integer):boolean;
     var aux:boolean;
        str1,str2:string;
    begin
        IgualTiempo:=((t1=t2)or
                        (((t1<minT)or(t1>maxT))and((t2<minT)or(t2>maxT))));
    end;

 Function HayPred(H:hojas;n:Ptero;var vlr:integer):boolean;
    Var ind:hojas;
        aux:nodos;
        sal:boolean;
 begin
    ind:=h;
    sal:=false;
    while ((ind<>nil)and(Not sal)) do
        begin
            aux:=ind^.leafs;
            while ((aux<>nil) and(aux^.nodo<>n)) do
                aux:=aux^.sig;
            if ((aux<>nil)and(aux^.nodo=n))
                then begin
                    sal:=true;
                    vlr:=ind^.valor;
                    end;
            ind:=ind^.sig;
        end;
    HayPred:=sal;
 end;
```

```
Procedure AgregarPred(Var H:hojas;n:ptero;value:integer);
   var nue:nodos;
       aux:hojas;
   begin
       new(nue);
       nue^.nodo:=n;
       aux:=h;
       while ((aux<>nil)and(aux^.valor<>value)) do
           aux:=aux^.sig;
       if((aux<>nil)and(aux^.valor=value))
           then nue^.sig:=aux^.leafs
           else begin
               nue^.sig:=nil;
               new(aux);
               aux^.valor:=value;
               aux^.sig:=h;
               h:=aux;
               end;
       aux^.leafs:=nue
   end;

{-------------------------------------------------------------------------------------------------}

Procedure AgregarArbol(A1:arbol);
 var aux:parb;
 begin
   new(aux);
   aux^.nodo:=A1;
   aux^.sig:=pri;
   pri:=aux;
 end;

Function newnodo(form,j,asg:string;val:boolean):ptero;
 var p:ptero;
 begin
   new(p);
   p^.formula:=form;
   p^.i:=j;
   p^.V:=val;
```

```
    p^.asg:=asg;
    p^.hijos:=nil;
    p^.marcado:=false;
    P^.procesado:=false;
    newnodo:=p
end;

Function crear(formula,i:string):arbol;
 var
    arb:arbol;
    k:integer;
    j:TPredTD;
    n1:ptero;
 begin
    new(arb);
    arb^.asgTD:=nil;
    for j:=minPredTD to maxPredTD do
        begin
            arb^.pe[j]:=nil;
            arb^.pe_[j]:=nil;
            for k:=minT to maxT do
                begin
                    arb^.pi[j,k]:=nil;
                    arb^.pi_[j,k]:=nil;
                end;
         end;
    new(arb^.raiz);
    arb^.raiz:=newnodo(formula,i,'',true);
    crear:=arb ;
 end;

Procedure newhijo(var n1,n2:ptero);
 var p:nodos;
 begin
    if n1=nil
        then n1:=n2
        else begin
            new(p);
            p^.sig:=n1^.hijos;
            p^.nodo:=n2;
            n1^.hijos:=p;
```

```
            end;
 end;

Procedure AgregarAsgTD(A:arbol;vrbl:char;vlr,i:integer);
 var aux:asigTD;
 begin
    new(aux);
    aux^.variable:=vrbl;
    aux^.valor:=vlr;
    aux^.i:=i;
    aux^.sig:=A^.asgTD;
    A^.asgTD:=aux;
 end;

Function HayAsgTD(A:arbol;VTD:char;j:integer;Var Value:integer):boolean;
var punt:asigTD;
    sal,afuera:boolean;
begin
    punt:=A^.asgTD;
    sal:=false;
    afuera:=((j<minT) or (j>maxT));
    while (punt<>nil) and Not(sal) do
        with punt^ do
            if (variable=VTD)and((i=j)or(afuera and((i<minT)or(i>maxT)) ))
                then sal:=true
                else punt:=punt^.sig;
    if sal
        then begin
            Value:=punt^.valor;
            HayAsgTD:=true
            end
     else begin
            Value:=0;
            HayAsgTD:=false
            end;
 end;

Procedure CopiarArbol(A1:arbol;Var A2:arbol;n1:ptero;Var n2:ptero);
 var k:integer;
    j:TPredTD;
    narb:ptero;
```

```
Procedure CopiarAsgTD(A1,A2:arbol);
 var punt,ult,aux:asigTD;
 begin
    if A1^.asgTD=nil
       then A2^.asgTD:=nil
       else begin
          punt:=A1^.asgTD;
          new(A2^.asgTD);
          A2^.asgTD^.variable:=punt^.variable;
          A2^.asgTD^.valor:=punt^.valor;
          A2^.asgTD^.i:=punt^.i;
          A2^.asgTD^.sig:=nil;
          ult:=A2^.asgTD;
          punt:=punt^.sig;
          while (punt<>nil)  do
             begin
                new(aux);
                aux^.variable:=punt^.variable;
                aux^.valor:=punt^.valor;
                aux^.i:=punt^.i;
                aux^.sig:=nil;
                ult^.sig:=aux;
                ult:=aux;
                punt:=punt^.sig;
             end;
          end;
 end;

Procedure CopiarNodo(na,p2:ptero);
 var nb:ptero;
    k,vlr:integer;
    j:TPredTD;
    proc:boolean;
    hij:nodos;

begin
    if na<>nil
       then with na^ do
          begin
             new(nb);
```

```
            nb^.formula:=formula;
            nb^.i:=i;
            nb^.V:=V;
            nb^.asg:=asg;
            nb^.hijos:=nil;
            nb^.marcado:=marcado;
            nb^.procesado:=procesado;
            if na=n1
               then n2:=nb;
            if na^.hijos=nil {es una hoja}
               then for j:=minPredTD to maxPredTD do
                  begin
                     if HayPred(A1^.pe[j],na,vlr)
                        then AgregarPred(A2^.pe[j],nb,vlr)
                        else if HayPred(A1^.pe_[j],na,vlr)
                           then AgregarPred(A2^.pe_[j],nb,vlr);
                     for k:=minT to maxT do
                        if HayPred(A1^.pi[j,k],na,vlr)
                           then AgregarPred(A2^.pi[j,k],nb,vlr)
                           else if HayPred(A1^.pi_[j,k],na,vlr)
                              then AgregarPred(A2^.pi_[j,k],nb,vlr);
                  end;
            if p2=nil
               then A2^.raiz:=nb
               else newHijo(p2,nb);
            hij:=hijos;
            while hij<>nil do
               begin
                  copiarNodo(hij^.nodo,nb);
                  hij:=hij^.sig;
               end;
         end
end;

begin
   new(A2);
   CopiarAsgTD(A1,A2);
   for j:=minPredTD to maxPredTD do
      begin
         A2^.pe[j]:=nil;
         A2^.pe_[j]:=nil;
```

```
            for k:=minT to maxT do
                begin
                    A2^.pi[j,k]:=nil;
                    A2^.pi_[j,k]:=nil;
                end;
        end;
    copiarNodo(A1^.raiz,nil)
end;

Function AgregarPi(cual:integer;A:arbol;n:ptero;p:string;i:integer;vlr:integer):boolean;
 var indice:TPredTD;
    ind:hojas;
    cualPi,cualNoPi:Hojas;
    sal:boolean;
 begin
    if p='C'
        then   indice:=C
        else   indice:=D;
    if cual=1
        then begin
            cualPI:=A^.Pi[indice,i];
            cualNoPi:=A^.Pi_[indice,i];
            end
        else begin
            cualPI:=A^.Pi_[indice,i];
            cualNoPi:=A^.Pi[indice,i];
            end;
        ind:=cualNoPI;
        sal:=false;
        while ((ind<>nil) and (Not(sal))) do
            begin
                if ind^.valor=vlr
                    then sal:=true;
                ind:=ind^.sig;
            end;
        if sal
            then AgregarPi:=false
            else begin
                AgregarPred(cualPI,n,vlr);
                if cual=1
                    then A^.Pi[indice,i]:=cualPI
```

```
                else A^.Pi_[indice,i]:=cualPI;
            AgregarPi:=true;
        end;
    end;

Function AgregarPe(cual:integer;A:arbol;n:ptero;p:string;vlr:integer):boolean;
 var indice:TPredTD;
    ind:hojas;
    cualPi,cualNoPi:Hojas;
    sal:boolean;
 begin
    if p='C'
        then   indice:=C
        else   indice:=D;
    if cual=1
        then begin
            cualPI:=A^.Pe[indice];
            cualNoPi:=A^.Pe_[indice];
            end
        else begin
            cualPI:=A^.Pe_[indice];
            cualNoPi:=A^.Pe[indice];
            end;
    ind:=cualNoPI;
    sal:=false;
    while ( (ind<>nil) and (Not(sal))) do
        begin
            if ind^.valor=vlr
                then sal:=true;
            ind:=ind^.sig;
        end;
    if sal
        then AgregarPe:=false
        else begin
            AgregarPred(cualPI,n,vlr);
            if cual=1
                then A^.Pe[indice]:=cualPI
                else A^.Pe_[indice]:=cualPI;
            AgregarPe:=true;
            end;
end;
```

```
Procedure SacarPi(cual:integer;A:arbol;p:string;i,vlr:integer);
 var indice:TPredTD;
    ant,aux,cualPI:hojas;
    nod:nodos;
 begin
    if p='C'
       then   indice:=C
       else   indice:=D;
    if cual=1
       then begin
          cualPI:=A^.Pi[indice,i];
          sacadoPI[indice,i]:=sacadoPI[indice,i]+[vlr];
          end
       else begin
          cualPI:=A^.Pi_[indice,i];
          sacadoPI_[indice,i]:=sacadoPI_[indice,i]+[vlr];
          end;
    aux:=cualPI;
    ant:=cualPI;
    while (aux<>nil) do
       begin
          if aux^.valor=vlr
             then begin
                nod:=aux^.leafs;
                while (nod<>nil) do
                   begin
                      nod^.nodo^.marcado:=true;
                      nod:=nod^.sig;
                   end;
                if aux=cualPI
                   then if cual=1
                           then A^.Pi[indice,i]:=aux^.sig
                           else A^.Pi_[indice,i]:=aux^.sig
                   else ant^.sig:=aux^.sig;
                dispose(aux);
                aux:=nil;
                end
             else begin
                ant:=aux;
                aux:=ant^.sig
```

```
                end;
    end;
end;

Procedure SacarPe(cual:integer;A:arbol;p:string;vlr:integer);
 var indice:TPredTD;
    ant,aux,cualPl:hojas;
    nod:nodos;
 begin
    if p='C'
       then  indice:=C
       else  indice:=D;
    if cual=1
       then begin
          cualPl:=A^.Pe[indice];
          sacadoPe[indice]:=sacadoPe[indice]+[vlr];
          end
       else begin
          cualPl:=A^.Pe_[indice];
          sacadoPe_[indice]:=sacadoPe_[indice]+[vlr];
          end;
    aux:=cualPl;
    ant:=cualPl;
    while (aux<>nil) do
       begin
          if aux^.valor=vlr
             then begin
                nod:=aux^.leafs;
                while (nod<>nil) do
                   begin
                      nod^.nodo^.marcado:=true;
                      nod:=nod^.sig;
                   end;
                if aux=cualPl
                   then if cual=1
                           then A^.Pe[indice]:=aux^.sig
                           else A^.Pe_[indice]:=aux^.sig
                   else ant^.sig:=aux^.sig;
                dispose(aux);
                aux:=nil;
                end
```

```
            else begin
                ant:=aux;
                aux:=ant^.sig
                end;
    end;
end;

Function FueSacadoDePi(cual:integer;p:string;i:integer;vlr:integer):boolean;
 var indice:TPredTD;
 begin
   if p='C'
      then  indice:=C
      else  indice:=D;
   if cual=1
      then FueSacadoDePi:=  (vlr IN sacadoPi[indice,i])
      else FueSacadoDePi:=(vlr IN sacadoPi_[indice,i]);
end;

Function FueSacadoDePe(cual:integer;p:string;vlr:integer):boolean;
 var indice:TPredTD;
 begin
   if p='C'
      then  indice:=C
      else  indice:=D;
   if cual=1
      then FueSacadoDePe:=  (vlr IN sacadoPe[indice])
      else FueSacadoDePe:=(vlr IN sacadoPe_[indice]);
end;

Procedure verArbol(A:arbol);
 var sangria:string;
   k:integer;
   aux:hojas;
   j:TPredTD;
   auxchar:char;
   puntasg:asigTD;

Procedure VerasigPred(str:string;H:hojas);
 begin
   if h<>nil
      then begin
```

```
            write(str,k,'(',auxchar,')={');
            while (h<>nil) do
                begin
                    write(H^.valor,' ');
                    h:=h^.sig;
                end;
                writeln('}');
            end;
end;

Procedure VerasigPe(str:string;H:hojas);
 begin
    if h<>nil
        then begin
            write(str,'(',auxchar,')={');
            while (h<>nil) do
                begin
                    write(H^.valor,' ');
                    h:=h^.sig;
                end;
            writeln('}');
            end;
end;

Procedure vernodo(p:ptero;sang:string);
 var k:integer;
    punt:nodos;
 begin
    write(sang,'____ ',p^.formula);
    write(' , ',p^.i);
    write(' , ',p^.V);
    if p^.asg=''
        then write(' , {}')
        else write(' , ',p^.asg);
    if p^.marcado
        then writeln('  ×')
        else writeln('  ß');
    punt:=p^.hijos;
    while punt<>nil do
        begin
            if punt^.sig=nil
```

```
            then verNodo(punt^.nodo,sang+'   ')
            else verNodo(punt^.nodo,sang+' | ');
         punt:=punt^.sig;
      end;
end;

begin
   clrscr;
   writeln;
         writeln('================================        Á        R        B        O        L
==================================');
   writeln;
   if A^.asgTD<>nil
      then begin
         writeln ('Asignaciones TD');
         puntasg:=A^.asgTD;
         while puntasg<>nil do
            begin
               write(' ',puntasg^.variable,'=');
               write(puntasg^.valor);
               write(' en ',puntasg^.i,',');
               puntasg:=puntasg^.sig;
            end;
         writeln;
         end;
   for j:=minPredTD to maxPredTD do
      begin
         if j=C
            then auxchar:='C'
            else auxchar:='D';
      VerAsigPe(' Pe ',A^.pe[j]);
      VerAsigPe(' Pe"',A^.pe_[j]);
      for k:=minT to maxT do
         begin
            VerAsigPred(' P',A^.pi[j,k]);
            VerAsigPred(' P"',A^.pi_[j,k]);
         end;
      end;
   writeln;
   writeln ('  ---------------------------- N O D O S ----------------------------');
   writeln;
```

```
    vernodo(A^.raiz,'');
end;

  begin { Inicialización }
    for j:=minPredTD to maxPredTD do
      begin
        SacadoPe_[j]:=[];
        SacadoPe[j]:=[];
        for k:=minT to maxT do
          begin
            SacadoPi[j,k]:=[];
            SacadoPi_[j,k]:=[];
          end;
      end;
  end.
```

UNIDAD CONJUNTO

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}


UNIT conjunto;
{ Contiene el manejo de los conjuntos expresados por comprención.
Un conjunto es un string que tiene un valor, y/o un límite, y/o un
intervalo, todos separados por un blanco y los límites están expresados
con el signo <. }

INTERFACE

uses terminos{,structur},frame;  { Diferencia con el de HC es frame en vez de structur}

Procedure ObtenerTemp(term,conj:string;Var inTemp,outTemp:string);
{ Dado el termino term se obtiene en inTemp y outTemp los valores de la
  variable infinita,  que pertenecen a asg, tal que el término term está
  dentro y fuera de Tf respectivamente.}

Function UnValor(conj:string):integer;
{ Dado un conjunto, devuelve un valor del mismo}

Procedure AgregarUnValor(Var conj:string; variable:char;valor:integer);
{ Agrega a un conjunto el valor dado }

Function HayySacarIntervalo(Var conj:string ;variable:char; Var minlim,maxlim:integer):boolean;
{Se fija si hay un intervalo en conj y lo saca }

Function LimiteInf(conj:string ; Var minlim:integer):boolean;
{ Devuelve True si hay algún límite inferior en conj; y en minlim devuelve
 su valor}

Function LimiteSup(conj:string ; Var maxlim:integer):boolean;
{ Devuelve True si hay algún límite usperior en conj; y en maxlim devuelve
 su valor}

Function MasDeUnElem(conj:string):boolean;
{Devuelve True si hay más de un elemento en conj}
```

```
Function Vacio(conj:string):boolean;
{Devuelve false si hay al menos un elemento en conj, y true en caso contrario}

Procedure AgregarIntervalo(Var conj:string;variable:char;min,max:integer);
{Se fija si hay algún intervalo en conj para la variable y devuelve en min
 y max sus valores límites}

Function Union(conj1,conj2:string):string;
{ Dado los conjuntos str1 y str2 devuelve su unión}

Function Interseccion(conj1,conj2:string):string;
{ Dado los conjuntos str1 y str2 devuelve su intersección}


IMPLEMENTATION


{---------- PROCEDIMIENTOS AUXILIARES ----------}

Function minVal(v1,v2:integer):integer;
begin
   if v1<=v2
      then minval:=v1
      else minval:=v2;
end;

Function maxVal(v1,v2:integer):integer;
begin
   if v1>=v2
      then maxval:=v1
      else maxval:=v2;
end;

{-----------------------------------------------------------------}


Function UnValor(conj:string):integer;
var str1:string;
   ind,valor1,code:integer;
begin
```

```
if conj<>''
   then begin
      ind:=1;
      while ((ind<=length(conj)) and (Not(ord(conj[ind])in[48..57]))) do
         ind:=ind+1;
      str1:='';
      while ((ind<=length(conj)) and(ord(conj[ind])in[48..57])) do
         begin
            str1:=str1+conj[ind];
            ind:=ind+1;
         end;
      val(str1,valor1,code);
      if ((ind=length(conj)+1) or (conj[ind]<>'<'))
         then UnValor:=valor1-1     { es un máximo }
         else UnValor:=valor1+1;       { es un mínimo o un intervalo}
      end;
end;


Function MasDeUnElem(conj:string):boolean;
 var str1,str2,str3:string;
   ind,valor1,valor2,code,auxtot:integer;
   auxbool:boolean;
 begin
   if conj=''
      then masDeUnElem:=false
      else begin
         auxbool:=false;
         auxtot:=0;
         ind:=1;
         while ((ind<=length(conj))and(Not auxbool)) do
            begin
               while ((ind<=length(conj)) and (Not(ord(conj[ind])in[48..57]))) do
                  ind:=ind+1;
               str2:='';
               while ((ind<=length(conj)) and(ord(conj[ind])in[48..57])) do
                  begin
                     str2:=str2+conj[ind];
                     ind:=ind+1;
                  end;
               val(str2,valor1,code);
```

```
                if ((ind=length(conj)+1)or(conj[ind]<>'<')or(ind+2>=length(conj))or(conj[ind+2]<>'<'))
                    then  auxbool:=true          { es un máximo o un mínimo }
                    else begin    { es un intervalo }
                        str3:='';
                        ind:=ind+3;
                        while ((ind<=length(conj)) and(ord(conj[ind])in[48..57])) do
                            begin
                                str3:=str3+conj[ind];
                                ind:=ind+1;
                             end;
                        val(str3,valor2,code);
                        if (valor2-valor1-1+auxtot)>1
                            then auxbool:=true
                            else auxtot:=auxtot+valor2-valor1-1;
                        end;
            end;
          MasDeUnElem:=auxbool;
          end;
end;


Function Vacio(conj:string):boolean;
 var str1,str2,str3:string;
    ind,valor1,valor2,code,auxtot:integer;
    auxbool:boolean;
 begin
    if conj=''
        then vacio:=false
        else begin
            auxbool:=true;
            ind:=1;
            while ((ind<=length(conj))and(auxbool)) do
                begin
                    while ((ind<=length(conj)) and (Not(ord(conj[ind])in[48..57]))) do
                    ind:=ind+1;
                    str2:='';
                    while ((ind<=length(conj)) and(ord(conj[ind])in[48..57])) do
                        begin
                            str2:=str2+conj[ind];
                            ind:=ind+1;
                        end;
```

```
                val(str2,valor1,code);
                if ((ind=length(conj)+1)or(conj[ind]<>'<')or(ind+2>=length(conj))or(conj[ind+2]<>'<'))
                   then  auxbool:=false          { es un máximo o un mínimo }
                   else begin    { es un intervalo }
                      str3:='';
                      ind:=ind+3;
                      while ((ind<=length(conj)) and(ord(conj[ind])in[48..57])) do
                         begin
                            str3:=str3+conj[ind];
                            ind:=ind+1;
                         end;
                      val(str3,valor2,code);
                      if (valor2-valor1-1)>=1
                         then auxbool:=false;
                      end;
             end;
          Vacio:=auxbool;
          end;
end;


Function HayySacarIntervalo(Var conj:string ;variable:char; Var minlim,maxlim:integer):boolean;
 var out2,str1,str2:string;
       ind,ind2,code:integer;
 begin
    out2:='<'+variable+'<';
    ind:=pos(out2,conj);
    if ind<>0
       then begin
          ind2:=ind+3;
          str1:='';
          while ((ind2<=length(conj)) and (ord(conj[ind2])in[48..57])) do
             begin
                str1:=str1+conj[ind2];
                ind2:=ind2+1;
             end;
          ind:=ind-1;
          str2:='';
          while ((ind>0) and (ord(conj[ind])in[48..57])) do
             begin
                str2:=conj[ind]+str2;
```

```
                ind:=ind-1;
            end;
        val(str2,minlim,code);
        val(str1,maxlim,code);
        delete(conj,ind+1,length(str1)+length(str2)+3);
        HayySacarIntervalo:=true;
        end
    else HayySacarIntervalo:=false;
end;

Function LimiteInf(conj:string ; Var minlim:integer):boolean;
 var auxstr:string;
   ind,ind2,code:integer;
   auxbool:boolean;
 begin
   ind:=pos('<',conj);
   auxbool:=false;
   while ind<>0 do
      if ((conj[ind+1]In VarInfinitas)and((length(conj)=ind+1)or(conj[ind+2]<>'<')))
         then begin
            auxstr:='';
            ind2:=ind-1;
            repeat
               auxstr:=conj[ind2]+auxstr;
               ind2:=ind2-1;
            until ((ind2=0)or(conj[ind2]=' '));
            val(auxstr,minlim,code);
            auxbool:=true;
            ind:=0;
            end
         else begin
            delete(conj,ind,2);
            ind:=pos('<',conj);
            end;
   LimiteInf:=auxbool;
end;


Function LimiteSup(conj:string ; Var maxlim:integer):boolean;
 var auxstr:string;
   ind,ind2,code:integer;
```

```
    auxbool:boolean;
 begin
    ind:=pos('<',conj);
    auxbool:=false;
    while ind<>0 do
        if (conj[ind-1]In VarInfinitas)
        then begin
            auxstr:='';
            ind2:=ind+1;
            repeat
                auxstr:=auxstr+conj[ind2];
                ind2:=ind2+1;
            until ((ind2=length(conj)+1) or Not(ord(conj[ind2])IN [48..57]));
            if ((ind2=length(conj)+1) or (conj[ind2]<>'<'))
                then begin
                    val(auxstr,maxlim,code);
                    auxbool:=true;
                    ind:=0;
                    end
                else begin
                    delete(conj,ind,2);
                    ind:=pos('<',conj);
                    end;
              end
        else begin
            delete(conj,ind,2);
            ind:=pos('<',conj);
            end;
    LimiteSup:=auxbool;
 end;


Procedure AgregarLimInf(Var conj:string;variable:char;min:integer);
 var str1,str2,str3:string;
    ind,valor1,valor2,code,long,long2:integer;
 begin
    str(min,str1);
    str1:=str1+'<'+variable;
    if conj=''
        then conj:=str1
        else begin
```

```
conj:=str1+' '+conj;
ind:=length(str1)+1;
while (ind<=length(conj)) do
   begin
      while ((ind<=length(conj)) and (Not(ord(conj[ind])In[48..57]))) do
         ind:=ind+1;
   str2:='';
   while ((ind<=length(conj)) and(ord(conj[ind])In[48..57])) do
      begin
         str2:=str2+conj[ind];
         ind:=ind+1;
      end;
   val(str2,valor1,code);
   if ((ind=length(conj)+1) or (conj[ind]<>'<'))
      then begin     { es un máximo }
         if (valor1>min)
            then conj:=Dominio(variable);
         end
      else if ((ind+2>=length(conj)) or (conj[ind+2]<>'<'))
         then        { es un mínimo }
            if (valor1> min)
               then begin
                  long:=length(str2)+3;
                  delete(conj,ind-1-length(str2),long);
                  ind:=ind-long;
                  end
               else begin
                  str(min,str1);
                  str1:=str1+'<'+variable+' ';
                  long:=length(str1);
                  delete(conj,pos(str1,conj),long);
                  ind:=ind-long;
                  min:=valor1;
                  end
         else begin    { es un intervalo }
            str3:='';
            ind:=ind+3;
            while ((ind<=length(conj)) and(ord(conj[ind])In[48..57])) do
               begin
                  str3:=str3+conj[ind];
                  ind:=ind+1;
```

```
                    end;
                val(str3,valor2,code);
                if min<valor1
                    then begin
                        long:=length(str3)+length(str2)+3;
                        delete(conj,ind-long,long);
                        ind:=ind-long;
                        end
                    else if min<valor2
                        then begin
                            long:=length(str3)+1;
                            delete(conj,ind-long,long);
                            str(min,str1);
                            str1:=str1+'<'+variable+' ';
                            long2:=length(str1);
                            delete(conj,pos(str1,conj),long2);
                            min:=valor1;
                            ind:=ind-long-long2;
                        end;
                end;
            end;
        end;
end;


Procedure AgregarLimSup(Var conj:string;variable:char;max:integer);
 var str1,str2,str3:string;
    ind,valor1,valor2,code,long,long2:integer;
 begin
    str(max,str1);
    str1:=variable+'<'+str1;
    if conj=''
        then conj:=str1
        else begin
            conj:=str1+' '+conj;
            ind:=length(str1)+1;
            while (ind<=length(conj)) do
                begin
                    while ((ind<=length(conj)) and (Not(ord(conj[ind])In[48..57]))) do
                        ind:=ind+1;
                    str2:='';
```

```
while ((ind<=length(conj)) and(ord(conj[ind])In[48..57])) do
   begin
      str2:=str2+conj[ind];
      ind:=ind+1;
   end;
val(str2,valor1,code);
if ((ind=length(conj)+1) or (conj[ind]<>'<'))
   then              { es un máximo }
      if(valor1< max)
         then begin
            long:=length(str2)+3;
            delete(conj,ind-long,long);
            ind:=ind-long;
            end
         else begin
            str(max,str1);
            str1:=variable+'<'+str1+' ';
            long:=length(str1);
            delete(conj,pos(str1,conj),long);
            ind:=ind-long;
            max:=valor1;
            end
   else if ((ind+2>=length(conj)) or (conj[ind+2]<>'<'))
      then if (valor1<max)  { es un mínimo }
            then conj:=Dominio(variable)
            else ind:=ind+2
      else begin   { es un intervalo }
         str3:='';
         ind:=ind+3;
         while ((ind<=length(conj)) and(ord(conj[ind])In[48..57])) do
            begin
               str3:=str3+conj[ind];
               ind:=ind+1;
            end;
         val(str3,valor2,code);
         if max>valor2
            then begin
               long:=length(str3)+length(str2)+3;
               delete(conj,ind-long,long);
               ind:=ind-long;
               end
```

```
                        else if max>valor1
                            then begin
                                long:=length(str2)+1;
                                delete(conj,ind-long-length(str3)-2,long);
                                str(max,str1);
                                str1:=variable+'<'+str1+' ';
                                long2:=length(str1);
                                delete(conj,pos(str1,conj),long2);
                                max:=valor2;
                                ind:=ind-long-long2;
                            end;
                    end;
                end;
            end;
end;


Procedure AgregarIntervalo(Var conj:string;variable:char;min,max:integer);
 var str1,str2,str3,str4:string;
    ind,valor1,valor2,long,code:integer;
    poner:boolean;
 begin
    str(min,str1);
    str(max,str4);
    if conj=''
        then begin
            str(min,str1);
            str(max,str4);
            conj:=str1+'<'+variable+'<'+str4;
            end
        else begin
            ind:=1;
            poner:=true;
            while ((ind<=length(conj)) and (poner)) do
                begin
                    while ((ind<=length(conj)) and (Not(ord(conj[ind])In[48..57]))) do
                        ind:=ind+1;
                    str2:='';
                    while ((ind<=length(conj)) and(ord(conj[ind])In[48..57])) do
                        begin
                            str2:=str2+conj[ind];
```

```
        ind:=ind+1;
    end;
val(str2,valor1,code);
if ((ind=length(conj)+1) or (conj[ind]<>'<'))
    then              { es un máximo }
        if (max<valor1)
            then poner:=false
            else begin
                if min<valor1
                    then begin
                        poner:=false;
                        AgregarLimSup(conj,variable,max);
                        end;
                    end
    else if ((ind+2>=length(conj)) or (conj[ind+2]<>'<'))
        then if (valor1<min)   { es un mínimo }
                then poner:=false
                else begin
                    if (valor1<max)
                        then begin
                            poner:=false;
                            AgregarLimInf(conj,variable,min);
                            end;
                    end
        else begin    { es un intervalo }
            str3:='';
            ind:=ind+3;
            while ((ind<=length(conj)) and(ord(conj[ind])in[48..57])) do
                begin
                    str3:=str3+conj[ind];
                    ind:=ind+1;
                end;
            val(str3,valor2,code);
            if (min<valor1)
                then begin
                    if (max>valor1)
                        then begin
                        long:=length(str2)+length(str3)+3;
                        delete(conj,ind-long,long+1);
                        str(max,str1);
                        ind:=ind-long-1;
```

```
                                    if (max<valor2)
                                        then max:=valor2;
                                    end;
                                end
                            else if (max<valor2)
                                then poner:=false
                                else
                                    if (min<valor2)
                                    then begin
                                        long:=length(str2)+length(str3)+3;
                                        delete(conj,ind-long,long+1);
                                        str(max,str1);
                                        ind:=ind-long-1;
                                        min:=valor1;
                                    end;
                        end;

            end;
        if poner
            then begin
                str(min,str1);
                str(max,str4);
                conj:=conj+' '+str1+'<'+variable+'<'+str4;
                end;
        end;
end;


Procedure AgregarUnValor(Var conj:string; variable:char;valor:integer);
begin
    AgregarIntervalo(conj,variable,valor-1,valor+1);
end;


Function Interseccion(conj1,conj2:string):string;
 var conj,str2,str3,auxconj:string;
    ind,valor1,valor2,code,min2,max2:integer;
    varInf:char;
 begin
    if (conj1='')or(conj2='')
        then Interseccion:=''
```

```
else begin
    ind:=1;
    conj:='';
    while (ind<=length(conj1)) do
        begin
        while ((ind<=length(conj1)) and (Not(ord(conj1[ind])In[48..57]))) do
            begin
                if conj1[ind] In varInfinitas
                    then varInf:=conj1[ind];
                ind:=ind+1;
            end;
        str2:='';
        while ((ind<=length(conj1)) and(ord(conj1[ind])In[48..57])) do
            begin
                str2:=str2+conj1[ind];
                ind:=ind+1;
            end;
        val(str2,valor1,code);
        if ((ind=length(conj1)+1) or (conj1[ind]<>'<'))
            then begin          { es un máximo }
                if LimiteSup(conj2,max2)
                    then AgregarLimSup(conj,varInf,minval(valor1,max2));
                if ((LimiteInf(conj2,min2))and(valor1>min2+1))
                    then AgregarIntervalo(conj,varInf,min2,valor1);
                auxconj:=conj2;
                while HayySacarIntervalo(auxconj,varInf,min2,max2) do
                    if valor1>min2+1
                        then AgregarIntervalo(conj,varInf,min2,minval(valor1,max2));
                end
            else begin
                varInf:=conj1[ind+1];
                if ((ind+2>=length(conj1)) or (conj1[ind+2]<>'<'))
                    then begin      {es un mínimo }
                        if (LimiteSup(conj2,max2)and(valor1<max2-1))
                            then AgregarIntervalo(conj,varInf,valor1,max2);
                        if LimiteInf(conj2,min2)
                            then AgregarLimInf(conj,varInf,maxval(min2,valor1));
                        auxconj:=conj2;
                        while HayySacarIntervalo(auxconj,varInf,min2,max2) do
                            if valor1<max2-1
                                then AgregarIntervalo(conj,varInf,maxval(min2,valor1),max2);
```

```
                        end
                    else begin   { es un intervalo }
                        str3:='';
                        ind:=ind+3;
                        while ((ind<=length(conj1)) and(ord(conj1[ind])In[48..57])) do
                            begin
                                str3:=str3+conj1[ind];
                                ind:=ind+1;
                            end;
                        val(str3,valor2,code);
                        if (LimiteSup(conj2,max2)and(valor1<max2-1))
                            then AgregarIntervalo(conj,varInf,valor1,minval(valor2,max2));
                        if (LimiteInf(conj2,min2)and(valor2>min2+1))
                            then AgregarIntervalo(conj,varInf,maxval(min2,valor1),valor2);
                        auxconj:=conj2;
                        while HayySacarIntervalo(auxconj,varInf,min2,max2) do
                            if ((valor1<max2-1)and(valor2>min2+1))
                                then
AgregarIntervalo(conj,varInf,maxval(min2,valor1),minval(max2,valor2));
                        end;
                end;
            end;
        interseccion:=conj;
        end;
end;


Function Union(conj1,conj2:string):string;
 var str2,str3:string;
    ind,valor1,valor2,code:integer;
    varInf:char;
{ Supongo que los dos conjuntos limitan a la misma variable }
 begin
    if (conj1='')or(conj2='')
        then Union:=conj1+conj2
        else begin
            ind:=1;
            while (ind<=length(conj1)) do
                begin
                    while ((ind<=length(conj1)) and (Not(ord(conj1[ind])In[48..57]))) do
                        begin
```

```
                    if conj1[ind] In varInfinitas
                        then varInf:=conj1[ind];
                    ind:=ind+1;
                end;
            str2:='';
            while ((ind<=length(conj1)) and(ord(conj1[ind])In[48..57])) do
                begin
                    str2:=str2+conj1[ind];
                    ind:=ind+1;
                end;
            val(str2,valor1,code);
            if ((ind=length(conj1)+1) or (conj1[ind]<>'<'))
                then AgregarLimSup(conj2,varInf,valor1) { es un máximo }
                else begin
                    varInf:=conj1[ind+1];
                    if ((ind+2>=length(conj1)) or (conj1[ind+2]<>'<'))
                        then begin        {es un mínimo }
                            AgregarLimInf(conj2,varInf,valor1);
                            ind:=ind+2
                            end
                        else begin    { es un intervalo }
                            str3:='';
                            ind:=ind+3;
                            while ((ind<=length(conj1)) and(ord(conj1[ind])In[48..57])) do
                                begin
                                    str3:=str3+conj1[ind];
                                    ind:=ind+1;
                                end;
                            val(str3,valor2,code);
                            AgregarIntervalo(conj2,varInf,valor1,valor2);
                            end
                    end;
            end;
        Union:=conj2;
        end;
end;


Procedure ObtenerTemp(term,conj:string;Var inTemp,outTemp:string);
var str1,str2,out2:string;
    varinf:char;
```

```
    aux1,aux2,max,min,maxlim,minlim,maxTemp,minTemp,ind:integer;
begin
    inTemp:='';
    OutTemp:='';
    if HayVarInf(term,varInf)
        then begin
        { ----- para saber si la var infinita está pos o neg.-----}
            aux1:=EvaluarTermino(replace(term,varInf,'0'));
            aux2:=EvaluarTermino(replace(term,varInf,'1'));
            if aux2>=aux1 {está positiva}
                then begin
                    max:=MaxT-aux1;
                    min:=MinT-aux1;
                    end
                else begin
                    max:=aux1-minT;
                    min:=aux1-maxT;
                    end;
            if limiteInf(conj,minlim)
                then if minlim < max
                        then begin
                            maxTemp:=max+1;
                            AgregarLimInf(OutTemp,varInf,max);
                            if minlim+1>=min
                                then minTemp:=minlim
                                else begin
                                    minTemp:=min-1;
                                    AgregarIntervalo(OutTemp,varInf,minlim,min)
                                    end;
                            AgregarIntervalo(inTemp,varInf,minTemp,maxTemp);
                            end
                        else AgregarLimInf(OutTemp,varInf,minlim);
            if limitesup(conj,maxlim)
                then if maxlim>min
                        then begin
                            minTemp:=min-1;
                            AgregarLimSup(outTemp,varInf,min);
                            if maxlim-1<=max
                                then maxTemp:=maxlim
                                else begin
                                    maxTemp:=max+1;
```

```
                    AgregarIntervalo(outTemp,varInf,max,maxlim)
                    end;
                AgregarIntervalo(inTemp,varInf,minTemp,maxTemp);
                end
            else AgregarLimSup(OutTemp,varInf,maxlim);
        while HayySacarIntervalo(conj,varInf,minlim,maxlim)do
          if ((minlim<max) and (maxlim>min))
            then begin
              if minlim+1>=min
                then begin
                  minTemp:=minLim;
                  if maxlim>max+1
                    then begin
                      maxtemp:=max+1;
                      AgregarIntervalo(outTemp,varInf,max,maxlim);
                      end
                    else maxTemp:=maxlim;
                  end
                else begin
                  mintemp:=min-1;
                  AgregarIntervalo(outTemp,varInf,minlim,min);
                  if maxlim>max+1
                    then begin
                      maxtemp:=max+1;
                      AgregarIntervalo(outTemp,varInf,max,maxlim);
                      end
                    else maxTemp:=maxlim;
                  end;
              AgregarIntervalo(inTemp,varInf,minTemp,maxTemp);
              end
            else AgregarIntervalo(OutTemp,varInf,minlim,maxlim);
      end;
end;

end.
```

UNIDAD TERMINOS

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}

UNIT terminos;
{ Provee el manejo de los string que representan fórmulas y términos }

INTERFACE

Function replace (str1:string;c1:char;c2:string):string;
{ Devuelve el string str1, reemplazando todas las ocurrencias de c1 por c2}

Function EvaluarTermino(formula:string):integer;
{ Dado un término , devuelve su valor }

Procedure TomarTermino(formula:string; var term:string);
{ De un fórmula dada toma la primer subfórmula }

Procedure TomarTermDist(formula:string; var ter1,ter2:string);
{ De un fórmula que tiene un Dist devuelve en ter1 su subfórmula y en ter2 el término temporal}

Procedure terminosFAtomica(formula:string; var f,ter1,ter2:string);
{ De un fórmula atómica dada devuelve en f el predicado, y en ter1 y ter2 sus dos terminos,
    o devuelve vacio en ter2 si es unaria, o ambos vacios si no tiene terminos }

Procedure terminoFormula(formula:string; var f,ter:string);
{ De un fórmula atómica dada devuelve en f el predicado, y en ter su termino, o
    devuelve vacio en ter si la fórmula no tiene termino. }


IMPLEMENTATION

Function replace (str1:string;c1:char;c2:string):string;
 var p:integer;
 begin
   p:=pos(c1,str1);
   while p<>0 do
      begin
         delete(str1,p,1);
```

```
            insert(c2,str1,p);
            p:=pos(c1,str1);
        end;
    replace:=str1;
end;


Function EvaluarTermino(formula:string):integer;
 var ter,code,code2,i,long,AUX,MENOS:integer;
    oper:char;
 begin
    if formula<>''
        then begin
            aux:=0 ;
            oper:='+';
            repeat
              Val(formula,ter,code);
              if code=0
                  then begin
                      if oper='+'
                          then aux:=aux+ter
                          else aux:=aux-ter;
                      formula:='';
                      end
                  else if Not(ord(formula[1]) in [48..57])
                      then begin
                          i:=1;
                          menos:=0;
                          while Not(ord(formula[i]) in [48..57]) do
                              begin
                                  if formula[i]='-'
                                      then menos:=menos+1;
                                  i:=i+1;
                              end;
                          if (menos mod 2)<>0
                              then if oper='+'
                                  then oper:='-'
                                  else oper:='+';
                          formula:=(copy(formula,i,length(formula)))
                          end
                      else begin
```

```
                    Val(copy(formula,1,code-1),ter,code2);
                    if oper='+'
                       then aux:=aux+ter
                       else aux:=aux-ter;
                    oper:=formula[code];
                    formula:=(copy(formula,code+1,length(formula)))
                    end;

        until formula='';
        EvaluarTermino:=aux;
      end
    else EvaluarTermino:=0;
end;


Procedure TomarTermino(formula:string; var term:string);
var cant,j,longfor,posand,parent:integer;
begin
  posand:=pos('&',formula);
  parent:=pos('(',formula);
  if posand=0
     then term:=formula
     else begin
        if parent=0
           then parent:=256;
        if parent>posand
           then term:=copy(formula,1,posand-1)
           else begin
              cant:=1;
              j:=parent;
              repeat
                 j:=j+1;
                 case formula[j] of
                    '(': cant:=cant+1;
                    ')': cant:=cant-1;
                 end;
              until  (cant=0) ;
              term:=copy(formula,1,j);
              end;
        end;
end;
```

```
Procedure terminoFormula(formula:string; var f,ter:string);
 var primer:integer;
 begin
    primer:=pos('(',formula);
    if primer<>0
       then begin
          f:=copy(formula,1,primer-1);
          ter:=copy(formula,primer+1,length(formula)-primer-1);
          end
       else begin
          f:=formula;
          ter:='';
          end;
 end;


Procedure TomarTermDist(formula:string; var ter1,ter2:string);
 var cant,j,i:integer;
 begin
    ter1:='';
    ter2:='';
    j:=length(formula)-1;
    while (formula[j]<>',') do
       begin
          ter2:=formula[j]+ter2;
          j:=j-1;
       end;
    ter1:=copy(formula,3,j-3);
 end;


Procedure terminosFAtomica(formula:string; var f,ter1,ter2:string);
 var primer,coma:integer;
 begin
    primer:=pos('(',formula);
    if primer<>0
       then begin
          f:=copy(formula,1,primer-1);
          coma:=pos(',',formula);
```

```
        if coma<>0
            then begin
                ter1:=copy(formula,primer+1,coma-primer-1);
                ter2:=copy(formula,coma+1,length(formula)-coma-1);
                end
            else begin
                ter1:=copy(formula,primer+1,length(formula)-primer-1);
                ter2:='';
                end
        end
    else begin
        f:=formula;
        ter1:='';
        ter2:='';
        end;
end;


END.
```

Prototipo del algoritmo de History-Checking

En este capítulo se detallan los programas fuentes del algoritmo de History-Checking. Hay 2 ejemplos, los que están detallados en el capítulo 7, por lo que hay dos unidades structur. Además para que se vea la fórmula del ejemplo del capítulo 5 se ha modificado el programa principal, por lo que también hay 2.

Como se ha dicho anteriormente la unidad *Terminos* es la misma que para el algoritmo anterior y la unidad *Conjunto* solo se modifica la sentencia del USE, por lo que ambas unidades ya están detalladas en el capítulo precedente.

PROGRAMA PRINCIPAL H_CHECK1.

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$M 55849,8192}
{ Programa de History-Checking. (c) Isaac Carlos Turquie}

program H_Checking;
{   Ejemplo del Algoritmo de History-checking del informe entregado

Evalúa una formula dada con la interpretacion de las unit corresp.
  ATENCION: Las formulas se especificaran sin blancos y con los siguientes
    simbolos, utilizados como palabras reservadas, no pudiendo ser usados
    como nombres de variables, funcion o predicados:
                    # : cuantificador universal,
            & : And,
            - : Not,
            % : Dist,
                    () Separador
          Ademas la variable, al lado del cuantificador no lleva () }



 uses algoritm,nodos,wincrt;

 var form,i:string;
     raiz:ptero;
     m:boolean;
```

```
begin
    WindowTitle[0]:=' ';
    WindowTitle[1]:='A';
    WindowTitle[2]:='l';
    WindowTitle[3]:='g';
    WindowTitle[4]:='o';
    WindowTitle[5]:='r';
    WindowTitle[6]:='i';
    WindowTitle[7]:='t';
    WindowTitle[8]:='m';
    WindowTitle[9]:='o';
    WindowTitle[10]:=' ';
    WindowTitle[11]:='d';
    WindowTitle[12]:='e';
    WindowTitle[13]:=' ';
    WindowTitle[14]:='H';
    WindowTitle[15]:='i';
    WindowTitle[16]:='s';
    WindowTitle[17]:='t';
    WindowTitle[18]:='o';
    WindowTitle[19]:='r';
    WindowTitle[20]:='y';
    WindowTitle[21]:='-';
    WindowTitle[22]:='C';
    WindowTitle[23]:='h';
    WindowTitle[24]:='e';
    WindowTitle[25]:='c';
    WindowTitle[26]:='k';
    WindowTitle[27]:='i';
    WindowTitle[28]:='n';
    WindowTitle[29]:='g';
    WindowTitle[30]:='.';
    windowOrg.x:=0;
    windowOrg.y:=0;
    windowSize.x:=700;
    windowSize.y:=500;
    ScreenSize.x:=130;
    ScreenSize.y:=500;
    InitWinCrt;
    clrscr;
```

```
    writeln;
    writeln('          La Fórmula del ejemplo es : ');
    form:='#z(%(#x-(Igual(m,x)&sube&-%(Igual(m,x+1),1)),z))';
    writeln('   '+form);
    while (form<>'') do
    begin
       writeln;
       write('Ingrese el instante de tiempo en el que se va a evaluar : ');
       readln(i);
       raiz:=newnodo(form,i,'',true);
       Procesar(raiz);
       verArbol(raiz);
       writeln;
       writeln;
       if not(raiz^.marca)
         then writeln(' E X I T O , la fórmula se satisface en el instante '+i+'.')
         else writeln(' No se puede satisfacer la fórmula en el instante '+i+'.');
       writeln('Presione cualquier tecla');
       readkey;
       clrscr;
       writeln;
       writeln(' Las fórmulas se especificaran sin blancos y con los siguientes');
       writeln('símbolos, utilizados como palabras reservadas, no pudiendo ser usados');
       writeln('como nombres de variables o predicados:  ');
       writeln('          # : cuantificador universal,');
       writeln('          & : And,                  ');
       writeln('          - : Not,                 ');
       writeln('          % : Dist,                 ');
       writeln('          () Separador              ');
       writeln('      Además la variable, al lado del cuantificador no lleva () ');
       writeln;
       Writeln('Ingrese una Fórmula para evaluar (ENTER para terminar) : ');
       readln(form);
    end;
    donewincrt
end.
```

PROGRAMA PRINCIPAL H_CHECK2.

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$M 54199,8192}
{ Programa de History-Checking. (c) Isaac Carlos Turquie}

program H_Checking;
{   Algoritmo de History-checking

Evalua una formula dada con la interpretacion de las unit corresp.
  ATENCION: Las formulas se especificaran sin blancos y con los siguientes
    simbolos, utilizados como palabras reservadas, no pudiendo ser usados
    como nombres de variables, funcion o predicados:
                    # : cuantificador universal,
            & : And,
            - : Not,
            % : Dist,
                    () Separador
          Ademas la variable, al lado del cuantificador no lleva () }



 uses algoritm,nodos,wincrt;

 var form,i:string;
     raiz:ptero;
     m:boolean;


begin
    WindowTitle[0]:=' ';
    WindowTitle[1]:='A';
    WindowTitle[2]:='l';
    WindowTitle[3]:='g';
    WindowTitle[4]:='o';
    WindowTitle[5]:='r';
    WindowTitle[6]:='i';
    WindowTitle[7]:='t';
    WindowTitle[8]:='m';
    WindowTitle[9]:='o';
```

```
WindowTitle[10]:=' ';
WindowTitle[11]:='d';
WindowTitle[12]:='e';
WindowTitle[13]:=' ';
WindowTitle[14]:='H';
WindowTitle[15]:='i';
WindowTitle[16]:='s';
WindowTitle[17]:='t';
WindowTitle[18]:='o';
WindowTitle[19]:='r';
WindowTitle[20]:='y';
WindowTitle[21]:='-';
WindowTitle[22]:='C';
WindowTitle[23]:='h';
WindowTitle[24]:='e';
WindowTitle[25]:='c';
WindowTitle[26]:='k';
WindowTitle[27]:='i';
WindowTitle[28]:='n';
WindowTitle[29]:='g';
WindowTitle[30]:='.';
windowOrg.x:=0;
windowOrg.y:=0;
windowSize.x:=700;
windowSize.y:=500;
ScreenSize.x:=130;
ScreenSize.y:=500;
InitWinCrt;
clrscr;
writeln;
writeln(' Las fórmulas se especificaran sin blancos y con los siguientes');
writeln('símbolos, utilizados como palabras reservadas, no pudiendo ser usados');
writeln('como nombres de variables o predicados:  ');
writeln('        # : cuantificador universal,');
writeln('        & : And,                  ');
writeln('        - : Not,                 ');
writeln('        % : Dist,                 ');
writeln('        () Separador              ');
writeln('     Además la variable, al lado del cuantificador no lleva () ');
writeln;
Writeln('Ingrese una Fórmula para evaluar (ENTER para terminar) : ');
```

```
    readln(form);
    while (form<>'') do
    begin
       writeln;
       write('Ingrese el instante de tiempo en el que se va a evaluar : ');
       readln(i);
       raiz:=newnodo(form,i,'',true);
       Procesar(raiz);
       verArbol(raiz);
       writeln;
       writeln;
       if not(raiz^.marca)
         then writeln(' E X I T O , la fórmula se satisface en el instante '+i+'.')
         else writeln(' No se puede satisfacer la fórmula en el instante '+i+'.');
       writeln('Presione cualquier tecla');
       readkey;
       clrscr;
       writeln;
       writeln(' Las fórmulas se especificaran sin blancos y con los siguientes');
       writeln('símbolos, utilizados como palabras reservadas, no pudiendo ser usados');
       writeln('como nombres de variables o predicados:  ');
       writeln('          # : cuantificador universal,');
       writeln('          & : And,                  ');
       writeln('          - : Not,                 ');
       writeln('          % : Dist,                 ');
       writeln('          () Separador               ');
       writeln('      Además la variable, al lado del cuantificador no lleva () ');
       writeln;
       Writeln('Ingrese una Fórmula para evaluar (ENTER para terminar) : ');
       readln(form);
    end;
    donewincrt
end.
```

UNIDAD ALGORITM.

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X-,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}

UNIT Algoritm;
{ Ejecuta el algoritmo de History-Checking,
 es decir construye y evalua el árbol de subformulas}

INTERFACE

 uses nodos,terminos,structur,conjunto;

 Procedure Procesar(nod:ptero);


IMPLEMENTATION

Procedure Procesar(nod:ptero);
 var h1:ptero;
    form2,taux,t1,t2: string;

 Procedure ProcFAtomica(n1:ptero);
 var instante,error:integer;
    varTD1,varI:char;
    valorTD:integer;
    valorstr,valorI,form,t1:string;
 begin
    with n1^ do
        begin
{---------- Construcción ----------}
            if HayVarInf(i,varI)
                then begin
                    str(UnValor(asg),valorI);
                    i:=replace(i,varI,valorI);
                    end;
            instante:=EvaluarTermino(i);
            terminoFormula(formula,form,t1);
            if HayVarTD(t1,varTD1)
            then
```

```
            begin
              repeat
                valorTD:=ni(varTD1,instante);
                str(valorTD,valorstr);
                t1:=replace(t1,varTD1,valorstr);
              until Not(HayVarTD(t1,varTD1));
              formula:=form+'('+t1+')';
            end;
{-------------- Marcado -------------------------}
        if predicadoTI(form)
          then marca:= P(formula)
          else marca:= Pi(formula,instante);
        if (V)
          then marca:=Not marca;
      end;
  end;

 Procedure ProcAnd(F1,F2:string);
 var h1,h2:ptero;
   asgnod:string;
   marcanod,tieneinf:boolean;
   hij:punthijos;
   aux:char;
 begin
   with nod^ do
     begin
{-------------- Construcción ---------------------}
       h1:=newnodo(F1,i,asg,V);
       newhijo(nod,h1);
       h2:=newnodo(F2,i,asg,V);
       newhijo(nod,h2);
       Procesar(h1);
       Procesar(h2);
{-------------- Marcado -------------------------}
       hij:=hijos;
       tieneInf:=(HayVarInf(i,aux));
       if Not(V)
         then begin
           marcanod:=true;
           asgnod:='xxx';
           while (hij<>nil) do
```

```
                begin
                    marcanod:=marcanod and hij^.hijo^.marca;
                    if (tieneInf or HayvarInf(hij^.hijo^.formula,aux))
                        then if (not (hij^.hijo^.marca))
                            then if asgnod='xxx'
                                then asgnod:=hij^.hijo^.asg
                                else asgnod:=Union(asgnod,hij^.hijo^.asg);
                    hij:=hij^.sig;
                end;
            end
        else begin
            marcanod:=false;
            asgnod:='xxx';
            while ((hij<>nil)and (not marcanod)) do
                begin
                    marcanod:=marcanod or hij^.hijo^.marca;
                    if (tieneInf or (HayvarInf(hij^.hijo^.formula,aux)))
                        then if asgnod<>'xxx'
                            then asgnod:=Interseccion(asgnod,hij^.hijo^.asg)
                            else asgnod:=hij^.hijo^.asg;
                    hij:=hij^.sig;
                end;
            end;
        if ((not marcanod)and(asgnod<>'xxx'))
            then asg:=asgnod;
        marca:=marcanod;
    end;
end;


Procedure ProcPt(n1:ptero);
var form2,t1,valorstr,asgnod:string;
  indice,k:integer;
  hij:punthijos;
  h1:ptero;
  aux:char;
  marcanod,tieneinf:boolean;
begin
  with n1^ do
    begin
      If formula[3]='('
```

```
            then t1:=copy(formula,4,length(formula)-4)
            else t1:=copy(formula,3,255);
         if (formula[2]In varInfinitas)
            then begin
{---------- Construcción ----------}
               h1:=newnodo(t1,i,asg,v);
               newhijo(n1,h1);
               Procesar(h1);
{---------- Marcado ----------}
               if hijos^.hijo^.marca
                  then marca:=true
                  else if V
                                    { me fijo si está incluido}
                     then
marca:=(dominio(formula[2])<>Interseccion(Dominio(formula[2]),hijos^.hijo^.asg))
                     else marca:=Vacio(hijos^.hijo^.asg);
               end
            else begin
{---------- Construcción ----------}
               for indice:=min(formula[2]) to max(formula[2]) do
                  begin
                     str(indice,valorstr);
                     form2:=replace(t1,formula[2],valorstr);
                     h1:=newnodo(form2,i,asg,V);
                     newhijo(n1,h1);
                     Procesar(h1);
                  end;
{---------- Marcado ----------}
               hij:=hijos;
               tieneInf:=(HayvarInf(formula,aux))or(HayVarInf(i,aux));
               if Not(V)
               then begin
                  marcanod:=true;
                  asgnod:='xxx';
                  while (hij<>nil) do
                     begin
                        marcanod:=marcanod and hij^.hijo^.marca;
                        if ((tieneInf)and(not (hij^.hijo^.marca)))
                           then if asgnod='xxx'
                                 then asgnod:=hij^.hijo^.asg
                                 else asgnod:=Union(asgnod,hij^.hijo^.asg);
```

```
                    hij:=hij^.sig;
                end;
            end
        else begin
            marcanod:=false;
            asgnod:='xxx';
            while ((hij<>nil)and (not marcanod)) do
                begin
                    marcanod:=marcanod or hij^.hijo^.marca;
                    if (tieneinf)and(not marcanod)
                        then if asgnod<>'xxx'
                            then asgnod:=Interseccion(asgnod,hij^.hijo^.asg)
                            else asgnod:=hij^.hijo^.asg;
                    hij:=hij^.sig;
                end;
            end;
        if ((not marcanod)and(asgnod<>'xxx'))
                then asg:=asgnod;
        marca:=marcanod;
        end;
    end;
end;


Procedure ProcDist(n1:ptero);
var instante,value,minTemp,maxTemp:integer;
  varTD1,varI2,varI:char;
  valorTD:integer;
  asg1,strI,strTD,j,inTemp,outTemp,re,newInst,asgnod:string;
  hij:punthijos;
  aux:char;
  marcanod:boolean;
begin
  with n1^ do
    begin
      tomartermDist(formula,t1,t2);
  {----- Obtengo Si(T2)----}
      if HayVarInf(i,varI)
        then begin
          str(UnValor(asg),strI);
          j:=replace(i,varI,strI)
```

```
        end
    else j:=i;
instante:=EvaluarTermino(j);
While HayVarTD(t2,varTD1) do
    begin
        valorTD:=ni(varTD1,instante);
        str(valorTD,strTD);
        t2:=replace(t2,varTD1,strTD);
    end;
if HayVarInf(t2,varI2)
    then begin
        if t2[1]='-'
            then   t2:=i+t2
            else   t2:=i+'+'+t2;
        ObtenerTemp(t2,Dominio(varI2),inTemp,outTemp);
        while HayySacarIntervalo(inTemp,varI2,minTemp,maxTemp)do
            { InTemp solo puede tener intervalos }
            for value:=minTemp+1 to maxTemp-1 do
                begin
                    str(value,re);
                    NewInst:=replace(t2,varI2,re);
                    asg1:='';
                    AgregarUnValor(asg1,varI2,value);
                    h1:=newnodo(t1,NewInst,asg1,V);
                    newhijo(nod,h1);
                    procesar(h1);
                end;
        h1:=newnodo(t1,t2,OutTemp,V);
        newhijo(nod,h1);
        procesar(h1);
    end
else
    If MasdeUnElem(asg)
        then begin
            if t2[1]='-'
                then   t2:=i+t2
                else   t2:=i+'+'+t2;
            ObtenerTemp(t2,asg,inTemp,outTemp);
            while HayySacarIntervalo(inTemp,varI,minTemp,maxTemp)do
                for value:=minTemp+1 to maxTemp-1 do
                    begin
```

```
                        str(value,re);
                        NewInst:=replace(t2,varl,re);
                        asg1:='';
                        AgregarUnValor(asg1,varl,value);
                        h1:=newnodo(t1,NewInst,asg1,V);
                        newhijo(nod,h1);
                        Procesar(h1);
                        end;
                    h1:=newnodo(t1,t2,OutTemp,V);
                    newhijo(nod,h1);
                    Procesar(h1);
                end
                else begin
                    if t2[1]='-'
                        then  t2:=j+t2
                        else  t2:=j+'+'+t2;
                    h1:=newnodo(t1,t2,asg,V);
                    newhijo(nod,h1);
                    Procesar(h1);
                end;
{---------- Marcado ------------------------}
        hij:=hijos;
        marcanod:=true;
        asgnod:='';
        while (hij<>nil) do
            begin
                marcanod:=marcanod and hij^.hijo^.marca;
                if (not (hij^.hijo^.marca))
                    then asgnod:=Union(asgnod,hij^.hijo^.asg);
                hij:=hij^.sig;
            end;
        if not marcanod
            then asg:=asgnod;
        marca:=marcanod;
    end;
end;


begin { ---------- de Procesar ---------- }

  case nod^.formula[1] of
```

```
    '-':with nod^ do
        begin
          form2:=copy(formula,2,255);
          tomartermino(form2,taux);
          if length(taux)<length(form2)
             then begin   { 2 términos y el & }
                t1:='-'+taux;
                t2:=copy(formula,length(t1)+2,255);
                ProcAnd(t1,t2);
                end
          else begin  { 1 término y el - }
             t1:=copy(formula,2,255);
             h1:=newnodo(t1,i,asg,Not(V));
             newhijo(nod,h1);
             Procesar(h1);
{------------- Marcado -------------------------}
             marca:=hijos^.hijo^.marca;
             if not(marca)
                then asg:=hijos^.hijo^.asg;
             end;
        end;
    '#':with nod^ do
        begin
          form2:=copy(formula,3,length(formula)-2);
          tomartermino(form2,taux);
          if length(taux)<length(form2) then
            begin                    { 2 términos y el & }
              t1:=formula[1]+formula[2]+taux;
              t2:=copy(formula,length(t1)+2,255);
              ProcAnd(t1,t2);
            end
          else ProcPT(nod);          { 1 término y el # }
        end;
    '(':with nod^ do
        begin
          tomartermino(formula,taux);
          if length(taux)<length(formula)
             then begin              { 2 términos y el & }
                t1:=taux;
                t2:=copy(formula,length(t1)+2,255);
                ProcAnd(t1,t2);
```

```
                end
            else begin                  { 1 término entre parentesis }
                formula:=copy(taux,2,(length(taux)-2));
                Procesar(nod);
                end;
        end;
    '%':with nod^ do
        begin
            form2:=copy(formula,2,255);
            tomartermino(form2,taux);
            if length(taux)<length(form2)
                then begin              { 2 términos y el & }
                    t1:='%'+taux;
                    t2:=copy(formula,length(t1)+2,255);
                    ProcAnd(t1,t2);
                    end
                else ProcDist(nod)      { 1 término y el dist }
        end;
    else
        with nod^ do
        begin
            tomartermino(formula,taux);
            if length(taux)<length(formula)
                then begin              { 2 términos y el & }
                    t1:=taux;
                    t2:=copy(formula,length(t1)+2,255);
                    procAnd(t1,t2);
                    end
                else ProcFAtomica(nod); { fórmula atómica }
        end;
  end;
end;
end.
```

UNIDAD STRUCTUR, PARA EL EJEMPLO DEL ASCENSOR.

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}

{ Interpretación para el ejemplo de history-checking del capítulo 5.
   Las variables tienen dominios en los enteros.
   Las funciones y las constantes numéricas se asumen en sus valores estandar.
   Los dominios de las variables se expresan en las funciomes max y min}

UNIT structur;

INTERFACE

uses terminos;

const
    varTI=['x','z'];
    varTD=['m'];
    varInfinitas=['z'];
    {------- Dominio Temporal, valores máximos y mínimos. --------}
    minT=2;
    maxT=4;


Function PredicadoTI(formula:string):boolean;
{ Dada una formula devuelve true si es TI o false en caso contrario }

Function P(formula:string):boolean;
{ dada una formula TI y su termino devuelve el valor de verdad de la misma }

Function Pi(formula:string;i:integer):boolean;
{ dada una formula TD, un instante de tiempo y su termino devuelve el valor de verdad de la misma }

Function Max(variable:Char):integer;
{Devuelve el valor m ximo de la variable especificada}

Function Min(variable:Char):integer;
{Devuelve el valor m¡nimo de la variable especificada}
```

```
Function ni(VarTD:char;i:integer):integer;
{ Dada una variable TD y su instante de tiempo, devuelve su valor}

Function HayVarTD(t:string;Var vTD:char):boolean;
{ Se fija si hay alguna variable TD en t, y si existe devuelve la primera que
 encuentra en vTD }

Function HayVarInf(t:string;Var varInf:char):boolean;
{ Se fija si hay alguna variable infinita en t, y si existe devuelve la primera que
 encuentra en varInf }

Function Dominio(variable:char):string;
{ Devuelve el dominio de la variable especificada}


IMPLEMENTATION

 type
    TvarTD=(m);
 const
    minVarTD=m;
    maxVarTD=m;

 var
    niTD:ARRAY[TvarTD,minT..maxT+1] of byte;


Function Dominio(variable:char):string;
 begin
    dominio:='0<z';
 end;


Function PredicadoTI(formula:string):boolean;
 begin
    PredicadoTI:=(formula='Igual');
 end;


Function P(formula:string):boolean;
 var f,t1,t2:string;
```

```
begin
    terminosFAtomica(formula,f,t1,t2);
    p:=(evaluartermino(t1)=evaluarTermino(t2));
end;


Function Pi(formula:string;i:integer):boolean;
begin
   pi:= ((formula='Sube') and (i in [2,3]));
end;


Function Max(variable:Char):integer;
begin
   case variable of
      'x':Max:=2;
   end;
end;


Function Min(variable:Char):integer;
begin
   case variable of
      'x':Min:=1;
   end;
end;


Function ni(VarTD:char;i:integer):integer;
begin
    if ((i<minT)or(i>maxT))
       then i:=maxT+1;
    ni:=niTD[m,i];
end;


Function HayVarTD(t:string;Var vTD:char):boolean;
var j,long:integer;
begin
   j:=1;
   long:=length(t);
```

```
    while ((j<=long) and Not(t[j] in varTD)) do
        j:=j+1;
    if j<=long
        then begin
            vTD:=t[j];
            HayVarTD:=true;
            end
        else begin
            vTD:=' ';
            HayVarTD:=false;
            end
end;


Function HayVarInf(t:string;Var varInf:char):boolean;
 var j,long:integer;
 begin
    j:=1;
    long:=length(t);
    while ((j<=long) and Not(t[j] in varInfinitas)) do
        j:=j+1;
    if j<=long
        then begin
            varInf:=t[j];
            HayVarInf:=true;
        end
        else begin
                varInf:=' ';
                HayVarInf:=false;
            end
end;


begin  { Inicialización }

    niTD[m,2]:=1;
    niTD[m,3]:=1;
    niTD[m,4]:=2;
    niTD[m,5]:=2;

end.
```

UNIDAD STRUCTUR.

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}

{ Interpretación para el algoritmo de history-checking
    Las variables tienen dominios en los enteros.
    Las funciones y las constantes numéricas se asumen en sus valores estandar.
    Los dominios de las variables se expresan en las funciomes max y min}

UNIT structur;

INTERFACE

Uses terminos;

const

    varTI=['x','y','z','r'];
    varTD=['m','n'];
    varInfinitas=['z'];
    {------- Dominio Temporal, valores máximos y mínimos. --------}
    minT=2;
    maxT=25;


Function Dominio(variable:char):string;
 { Devuelve el dominio de la variable especificada}

Function PredicadoTI(formula:string):boolean;
{ Dada una formula devuelve true si es TI o false en caso contrario }

Function P(formula:string):boolean;
{ Dada una formula TI y su termino devuelve el valor de verdad de la misma }

Function Pi(formula:string;i:integer):boolean;
{ Dada una formula TD, un instante de tiempo y su termino devuelve el valor de verdad de la misma }

Function Max(variable:Char):integer;
{ Devuelve el valor m ximo de la variable especificada}
```

```
Function Min(variable:Char):integer;
{ Devuelve el valor m¡nimo de la variable especificada}

Function ni(VarTD:char;i:integer):byte;
{ Dada una variable TD y su instante de tiempo, devuelve su valor}

Function HayVarTD(t:string;Var vTD:char):boolean;
{ Se fija si hay alguna variable TD en t, y si existe devuelve la primera que
 encuentra en vTD }

Function HayVarInf(t:string;Var varInf:char):boolean;
{ Se fija si hay alguna variable infinita en t, y si existe devuelve la primera que
 encuentra en varInf }


IMPLEMENTATION

const
   PredTI=['A','B'] ;
   PredTD=['C','D'];

type
   TPredTI=(A,B) ;
   TPredTD=(C,D);
   TVarTD=(m,n);
   valores=set of byte;

const
   minPredTD=C;
   maxPredTD=D;
   minVarTD=m;
   maxVarTD=n;

var
   longT:integer; {  indica la longitud de T }
   niTD:ARRAY[TvarTD,minT..maxT+1] of byte;
   PiTD:array[TpredTD,minT..maxT+1] of valores;
   j:integer;
```

```
Function Dominio(variable:char):string;
 begin
    dominio:='0<z';
 end;


Function PredicadoTI(formula:string):boolean;
 begin
    PredicadoTI:=(formula[1] in predTI);
 end;

Function P(formula:string):boolean;
 var f,t1,t2:string;
 begin
    terminosFAtomica(formula,f,t1,t2);
    if f='A'
        then P:=(evaluartermino(t1) in [1,2,3,4,5,6])
        else P:=(evaluartermino(t1) in [23,24,25,89,125]);
 end;

Function Pi(formula:string;i:integer):boolean;
    var cual:TpredTD;
        f,t1,t2:string;
 begin
    if not((mint<=i) and (i<=maxT))
        then i:=maxT+1;
    terminosFAtomica(formula,f,t1,t2);
    if f='C'
        then cual:=C
        else cual:=D;
    pi:=(evaluartermino(t1) in PiTD[cual,i]);
 end;


Function Max(variable:Char):integer;
 begin
    case variable of
        'x':Max:=5;
        'y':Max:=4;
        'r':Max:=2;
        'z':Max:=7;
```

```
    end;
end;


Function Min(variable:Char):integer;
 begin
    case variable of
        'x':Min:=2;
        'y':Min:=1;
        'r':Min:=2;
        'z':Min:=0;
    end;
 end;


Function ni(VarTD:char;i:integer):byte;
 begin
    if not((mint<=i) and (i<=maxT))
        then i:=maxT+1;
    case varTD of
        'm':ni:=niTD[m,i];
        'n':ni:=niTD[n,i];
    end;
 end;


Function HayVarTD(t:string;Var vTD:char):boolean;
 var j,long:integer;
 begin
    j:=1;
    long:=length(t);
    while ((j<=long) and Not(t[j] in varTD)) do
        j:=j+1;
    if j<=long
        then begin
            vTD:=t[j];
            HayVarTD:=true;
        end
        else begin
            vTD:=' ';
            HayVarTD:=false;
```

```
            end
 end;


Function HayVarInf(t:string;Var varInf:char):boolean;
 var j,long:integer;
 begin
    j:=1;
    long:=length(t);
    while ((j<=long) and Not(t[j] in varInfinitas)) do
        j:=j+1;
    if j<=long
        then begin
            varInf:=t[j];
            HayVarInf:=true;
            end
        else begin
            varInf:=' ';
            HayVarInf:=false;
            end
 end;


begin   { Inicialización }
    for j:=minT to maxT do
        begin
            PiTD[C,j]:=[2,3,4,j-2];
            PiTD[D,j]:=[1,2,j+1];
            niTD[m,j]:=j+1;
            niTD[n,j]:=j-2;
        end;
    PiTD[C,maxT+1]:=[2,3,4];
    PiTD[D,maxT+1]:=[1,2];
    niTD[m,maxT+1]:=1;
    niTD[n,maxT+1]:=2;
 end.
```

UNIDAD NODOS.

```
{$B-,F+,G+,I+,K+,L+,N-,P-,Q+,S+,T-,V+,W+,X+,Y+}
{$C MOVEABLE DEMANDLOAD DISCARDABLE}

UNIT nodos;
{ Provee las funciones de los nodos del árbol del algoritmo    }

INTERFACE

uses Wincrt;

type
   ptero=^nodo;
   punthijos=^reghijo;
   reghijo= record
      hijo:ptero;
       sig:punthijos;
   end;
   nodo = record
      formula:string;
      i:string;
      V:boolean;
      asg:string;
      marca:boolean;
      hijos:punthijos;
   end;


Function newnodo(form,j,asg:string;V:boolean):ptero;
{ Crea un nuevo nodo y devuelve su ptero}

Procedure newhijo(var n1,n2:ptero);
{ Agrega un nodo hijo n2 al nodo n1}

Procedure verArbol(n:ptero);
{ Muestra el árbol en pantalla }


IMPLEMENTATION
```

```
Function newnodo(form,j,asg:string;V:boolean):ptero;
 var p:ptero;
 begin
    new(p);
    p^.formula:=form;
    p^.i:=j;
    p^.asg:=asg;
    p^.V:=V;
    p^.hijos:=nil;
    newnodo:=p
 end;


Procedure newhijo(var n1,n2:ptero);
 var p:punthijos;
 begin
    if n1=nil
        then n1:=n2
        else begin
            new(p);
            p^.sig:=n1^.hijos;
            p^.hijo:=n2;
            n1^.hijos:=p;
            end;
end;


Procedure verArbol(n:ptero);
 var sangria:string;
        k:integer;
        auxchar:char;

 Procedure vernodo(p:ptero;sang:string);
    var k:integer;
        punt:punthijos;
    begin
        write(sang,'____ ',p^.formula);
        write(' , ',p^.i);
        write(' , ',p^.V);
        if p^.asg='
```

```
            then write(' , {}')
            else write(' , ',p^.asg);
        if p^.marca
            then writeln('   ×')
            else writeln('   ß');
         punt:=p^.hijos;
         while punt<>nil do
            begin
               if punt^.sig=nil
                  then verNodo(punt^.hijo,sang+'   ')
                  else verNodo(punt^.hijo,sang+' | ');
               punt:=punt^.sig;
            end;
end;

    begin
        clrscr;
        writeln ('---------------------- A R B O L ---------------------------');
        WRITELN;
        writeln ('                N O D O S                ');
        vernodo(n,'');
    end;


end.
```